7-1-2016

AN INFORMATION MODEL IN THE DOMAIN OF DISASSEMBLY PLANNING FOR SUSTAINABLE MANUFACTURING

Bicheng Zhu
Syracuse University

ABSTRACT

Disassembly, a process of separating the End of Life (EOL) product into discrete components for re-utilizing their associated residual values, is an important part for the sustainable manufacturing. This work focuses on the modeling of the disassembly planning related information, and develops a Disassembly Information Model (DIM) based on an extensive investigation of various informational aspects of the disassembly planning. The developed Disassembly Information Model, which represents an appropriate systematization and classification of the products, processes, uncertainties and degradations related information, follows a layered modeling methodology. In this layered configuration, the DIM is subdivided into three distinct layers with an intent to separate general knowledge into different levels of abstractions, and to reach a balance between information reusability and information usability. The performance evaluation of the DIM (usability and reusability) is accessed by successful implementations of the DIM model into two prototype software applications in the domain of disassembly planning.

The first application, called the Disassembly Sequence Generator (DSG), identifies the optimal disassembly sequence using a CAD based searching algorithm and a disassembly Linear Programming (LP) model. The searching process results in an AND/OR graph, which represents all the feasible disassembly sequences of a specific EOL product; whereas the LP model takes the AND/OR graph as an input and determines the economically optimal process sequence among all the possibilities.

The second application is called the Adaptive Disassembly Planning (ADP), which further takes the EOL product uncertainty and degradation issues into consideration. In order to address these issues, fuzzy logic and Bayesian Network methodologies are used to develop a Disassembly Decision Network (DDN), which adaptively generates the optimal disassembly sequence based on the current available information.

This research work is the first attempt to develop a comprehensive Information Model in the domain of disassembly planning. The associated modeling methodology that has been developed in this research is generic and scalable, and it could be widely adopted in other engineering domains, like product assembly, production planning, etc. The ultimate objective of this work is to standardize the DIM into a reference model that will be acknowledged and agreed upon by the sustainable manufacturing community.

AN INFORMATION MODEL IN THE DOMAIN OF DISASSEMBLY PLANNING FOR SUSTAINABLE MANUFACTURING

A Dissertation

By

Bicheng Zhu

B.S., Shanghai Jiao Tong University, 2007
M.S., Syracuse University, 2012

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Mechanical and Aerospace Engineering

Syracuse University

July, 2016

ACKNOWLEDGMENTS

This dissertation represents not only my work at the keyboard, it is a milestone in more than four years of work in the Knowledge Engineering Laboratory at Syracuse University. Here, I wish to express my sincere appreciation to those who have contributed to this thesis and supported me in one way or the other during this amazing journey.

First of all, I am extremely grateful to my supervisor, Professor Utpal Roy, for his guidance and all the useful discussions and brainstorming sessions, especially during the difficult problem formulation stage. I also remain indebted for all the opportunities (NIST research visit/workshop, technical research conferences, classroom teaching practice, industrial company collaboration, etc.) that Dr. Roy provided for preparing me for a better researcher.

I would also like to thank my committee members, professor Young B. Moon, professor John F. Dannenhoffer, professor Jianshun Zhang, professor Pranav Soman and professor Riyad S. Aboutaha for serving as my committee members and providing all types of valuable suggestions regarding to this work.

A big "Thank you!" also goes out to all the lab colleagues and friends, including Heng Zhang, Yunpeng Li, Kai Sun, Hang Yin, Omer Yaman and Mehemet Ilteris Sarigecili, for all their useful suggestions but also for being there to listen when I needed an ear.

Words cannot express the feelings I have for my parents (Jiafeng Zhu and Demei Xu) for their constant unconditional support - both emotionally and financially. I would not be here if it not for you. Thank you for having faith on me during this challenging journey.

Finally, I would like to acknowledge the most important person in my life – my wife Yunxing Nian. She has been a constant source of strength and inspiration. There were times during the past four years when everything seemed hopeless and I didn't have any hope. It was her determination and constant encouragement that ultimately made it possible for me to see this project through to the end.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ADP	Adaptive Disassembly Planning
AsD	Assembly Design Ontology
BN	Bayesian Network
CADP	Computer Aided Disassembly Planning
CPM	Core Product Model
DAG	Directed Acyclic Graph
DAO	Design Activity Ontology
DDN	Disassembly Decision Network
DFM	Design for Manufacturing
DIM	Disassembly Information Model
DL	Description Logic
DPN	Disassembly Petri Net
DSG	Disassembly Sequence Generator
D-UN	Disassembly Object Function Uncertainty Node
D-UTN	Disassembly Object Utility Node
EOL	End of Life
IM	Information Model
IoT	Internet of Things
LCU	Life Cycle Unit
LP	Linear Programming
MASON	Manufacturing Semantic Ontology
MEU	Maximum Expected Utility
OAM	Open Assembly Model
OWL	Web Ontology Language
P-DN	Process Decision Node
PEID	Product Embedded Information Device
PLM	Product Lifecycle Management
P-UN	Process Uncertainty Node (P-UN)

P-UTN	Process Utility Node
RoHS	Restriction of Hazardous Substances Directive
SWRL	Semantic Web Rule Language
TR	Transition Arc
UML	Unified Modeling Language
UN	Uncertainty Node
URI	Unified Resources Identifier
UTN	Utility Node
W3C	World Wide Web Consortium
WEEE	Waste Electrical and Electronic Equipment

Chapter 1 INTRODUCTION

In this chapter, an overview of the research performed in this dissertation is presented. The chapter begins by providing the technical context and background of the research. As the main topic of this work, the concept of Information Model (IM) is further discussed in detail. The problem statement, research contributions, and research methodology are then addressed. Lastly, this chapter is wrapped up by outlining the structure of the overall dissertation.

1.1 Research Background

A succinct and comprehensive definition of End of Life (EOL) product is provided by the European Economic Community, which defines the EOL product as “any substance or object which the holder discards or intends or is required to discard” (Gharfalkar, Court, Campbell, Ali & Hillier, 2015). Normally, the discarded EOL product may or may not be totally obsolete, and a recovery process can be applied to restore the contained value as a form of energy, material or product. Such recovery processes have been more and more studied under the popular paradigm of sustainable manufacturing, which has the objective to carry out economically-sound manufacturing/de-manufacturing processes that maximize the possible profits and minimize negative environmental impacts by utilizing different recovery options, such as recycling, reuse, and remanufacturing.

On the other hand, governments have already started to impose regulatory obligations on manufacturing companies, which mandate manufacturers to set up plans for collection, recycling and recovery for specific types of products. For instance, the Waste Electrical and Electronic Equipment Directive is the European community’s directive on the Waste Electrical and Electronic Equipment (WEEE), which became European law in February, 2003. The Restriction of Hazardous Substances Directive (RoHS) was also adopted in February 2003 by the European

Union to restrict the use of certain hazardous substances in the electrical and electronic equipment. In the United States, 25 states have passed legislative regulations, mandating statewide electronic waste (e-waste) recycling and several more states are working on passing new laws or improving the existing laws. All laws, except those in California and Utah use the “Producer Responsibility” approach, where the manufacturers must pay for recycling. Also, 65% of the U.S. population has been covered by a certain state level e-waste recycling laws since 2003 (Millar, 2005).

Both the potential economic profits and the regulatory laws motivate the study of the EOL product recovery modeling and implementation. As indicated in figure 1.1, four major EOL product recovery paths, named recycling, remanufacturing, direct reuse and disposal, have been identified (indicated as a green ellipse). Even though these paths consider various recovery strategies, all of them involve some level of disassembly process. In this sense, carrying out the disassembly process “optimally” plays a critical role in the entire process of the EOL product recovery. Over the years, various methods ranging from network theory to mathematical programming have been applied in the domain of product disassembly (Dong & Arndt, 2003). Unfortunately, not much work has been reported regarding the information aspect of the disassembly problem, which in the author’s opinion, is the bottleneck of the current disassembly related research. In detail, the challenge is that disassembly planners have limited knowledge on what information is critical in the planning of the disassembly process, how to access this information, and, finally how to utilize the updated on-site information (which is unknown in the beginning of the disassembly process) for dynamically adapting the “optimal” disassembly process plan. Also, an EOL product is highly independent and has to be treated individually, which further aggravates the above mentioned problems.

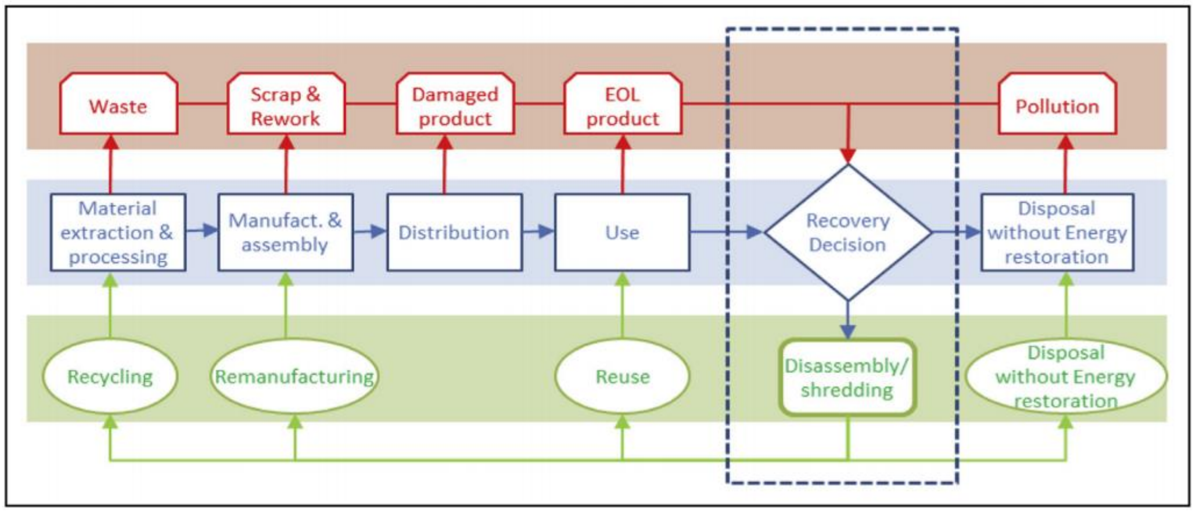


Figure 1.1: EOL Product Recovery Option (Ziout, 2013)

1.2 Information Modeling

Information Model, originates from software engineering and is a representation of concepts, relationships, constraints and rules for a chosen domain of discourse. It can provide a sharable, stable, and organized structure of information under some domain context (Halpin, 2001).

In the domain of manufacturing, a notable development in the IM field is the NIST's Core Product Model (CPM). It is a Unified Modeling Language (UML) based model intended to capture the full range of engineering information commonly shared in product development (Foufou, Fenves, Bock, Rachuri & Sriram, 2005). CPM focuses on modeling the general, common and generic product information, and excludes the information which is domain specific. NIST further developed another information model called "Open Assembly Model" (OAM) (Baysal, Roy, Sudarsan, Sriram & Lyons, 2004) which extends CPM. Along with the structural information, it represents the function, form, and behavior information related to an assembly, and defines a system level conceptual model. A comprehensive review on IM is presented in Chapter 2.

1.3 Goal of the Information Model Development

Generally, any Information Model has to meet two major goals: to be usable and to be reusable. IEEE Standard defines reusability as "the degree to which a software module or other work product can be used in more than one computing program or software system" (*IEEE standard glossary of software engineering terminology*.1983). Similarly, Information Model reusability can be defined as "the adaptation capability of an Information Model to arbitrary application contexts", including those contexts "that were not envisioned at the time of the creation of the Information Model" (Cysneiros, Werneck & Kushniruk, 2005). It should be understood that it is not feasible and desirable to develop an IM that is equally fitting to all application contexts (Borst, Akkermans & Top, 1997); rather the goal of reusability is to design an IM which can be extended and adapted to a large number of applications in the domain of interest.

On the other hand, usability denotes the degree to which the software component is useful for a specific task or application. By definition, an IM is rarely ready for use, but must always be adapted and refined to a knowledge base for the tentative application. Therefore, the goal of IM usability can be rephrased as minimizing "the effort required to customize the IM so that it can be used by humans or machines in a given application context" (Cysneiros, Werneck & Kushniruk, 2005).

As the reader might already have noticed, IM reusability and usability are contradicting each other: increasing the reusability of knowledge implies the maximization of using this knowledge among several kinds of tasks. The resulting IM would be general in nature; increasing usability implies providing all information related to a specific task and the resulting IM would have redundant information for other tasks and thus would not be appropriate. Consequently, it is difficult to simultaneously achieve high degrees of usability and reusability: Specializing in one kind of task makes the IM more useable for this particular task, but it also decreases the likelihood of its reusability; a highly abstract IM, on the other hand, may be applicable to a variety of different tasks, but it is unlikely to be proved useful for any of these tasks without extensive modification and detailing. This is known as the reusability-usability trade-off problem in the literature (Klinker, Bhola, Dallemagne, Marques & McDermott, 1991). In this research, a layered IM development methodology is developed to address this issue and it is presented in detail in Chapter 3.

1.4 Problem Statement

Based on the initial review of the background of the product recovery & disassembly and the Information Model, the overall research problem carried out in this dissertation can be summarized as follows:

Development and Implementation of a Disassembly Information Model (DIM) for efficient disassembly planning activities.

In detail, the research problem can be broken down to answer the following research questions:

Q1: What is the information required for disassembly planning and how to model it so that it can be both usable and reusable in the domain of disassembly planning (Modeling Methodology)?

Hypothesis: Disassembly related information can be identified and generalized through the literature reviews, and they can be partitioned into relevant sub models. A layered IM development methodology can address the reusability-usability trade-off problem. To address this hypothesis, the following objective is highlighted:

Objective: Identify the information requirements in the domain of disassembly planning and develop a Disassembly Information Model (DIM) which serves as a consensual information basis in the domain of disassembly. The developed DIM should provide basic information infrastructure, which can be proved to be both useful and reusable.

Q2: How to implement the disassembly information model?

Hypothesis: Description Logic (DL) based Web Ontology Language (OWL), as recommended by the World Wide Web Consortium (W3C) for the future semantic web, can be used to formally and computationally implement the DIM. To address this hypothesis, the following objective is highlighted:

Objective: Implementation of the Disassembly Information Model into formal Web Ontology Language (OWL) so that specific product information can be published and accessed through the web. Furthermore, semantic queries are developed for necessary information retrieval.

Q3: How to validate implemented Information Model?

Hypothesis: The performance of DIM (reusability-usability) can be partially validated through disassembly planning related application developments. To address this hypothesis, the following objective is highlighted:

Objective: Validating the whole DIM is a challenging process and there actually exists no formal IM validation process. In practice, it is common that an IM is upgraded and modified for improvement even after it has been published in the community. A good IM will be utilized by different applications in the targeted domain by extending itself to meet the application requirements. After years of such practices, it will be accepted by the domain community and promote itself to the standard level or the reference model level. Thus, in this work, we partially validate the usefulness and reusability of the DIM by developing of two disassembly planning applications, (1) Disassembly Sequence Generator and (2) Adaptive Disassembly Planning considering component and operation uncertainty, based on the DIM.

1.5 Research Contribution

To the author's knowledge, this work is a first attempt for the development & utilization of a comprehensive Information Model in the domain of disassembly planning, under the paradigm of sustainable manufacturing. Detailed contributions are broken down into the following aspects:

- Formal disassembly information representation. Most of the current studies on disassembly modeling are domain and algorithm specific, thus the information is isolated and heterogeneous. That's why information sharing is difficult. The developed DIM will be targeted at providing a formal, consensual information foundation, which can be promoted to a reference model in the future.
- DIM based disassembly planning application modeling. Most of the research works on Information Modeling are focusing on the development of IM structure, whereas, the application of IM in a real application task is lagging behind. This work fills in this gap by developing two disassembly planning applications based on extension of DIM: (1) Disassembly Sequence Generator and (2) Adaptive Disassembly Planning.
    - Disassembly Sequence Generator: DIM is extended for Disassembly Sequence Generator application, and a CAD based graph searching algorithm is developed to find all possible disassembly sequences of a specific EOL product. The detail of this application is presented in Chapter 4.
    - Adaptive Disassembly Planning: DIM is extended for the Adaptive Disassembly Planning. The fuzzy logic and Bayesian theorem are combined to handle the uncertainty issues both in the component quality (well-maintained or broken) and in the operation status (fail or success). The detail of this application is presented in Chapter 5.

1.6 Research Methodology and Thesis Outline

The overall research methodology of this work consists of five logical steps which includes: (1) a review of the current works and technologies (Chapter 2), (2) the development of new concepts and methodologies (Chapter 3), (3) the implementation and testing of the developed concepts and methods (Chapter 3, 4 & 5), (4) the overall evaluation of the results (Chapter 3, 4 & 5), and (5) the possible future extensions of this work (Chapter 6). The detail information about each step is described below:

Chapter 2 reviews the scientific background and establishes the terminologies required for discussing the development and utilization of DIM, thus providing the basis for the subsequent chapters.

It starts off by reviewing the research domain of product disassembly. After a systematic study, we found that although much work has been done in recent years, a systematic and integrated Information Model for various aspects of disassembly planning application has never been formed as a coherent body of knowledge. Next, concepts of IM are presented: We first contrast the similar but different perceptions of IM in the areas of philosophy and computer science. Next, the specification of IM through informal and formal languages is discussed; the latter option is further elaborated by describing the modeling capabilities of formal ontology languages. Then, we wrap up the discussion of IM by reviewing the existing developments of IM in the domain of manufacturing.

Chapter 3 comprehensively presents the development of DIM. The chapter starts with the information requirement analysis in the domain of disassembly planning, which results in a high level domain conceptualization. Next, a layered IM modeling methodology is proposed, with the intention to find a modeling balance between IM reusability and usability. Detailed DIM model is introduced afterwards, starting from the abstract models like “N-ary relationship Model” and “Graph Model” in the upper layer to the specific models like “Disassembly Sequence Generator” and “Adaptive Disassembly Planning” in the bottom layer (Figure 1.2). This chapter ends with the formal OWL implementation of the DIM.

Chapter 4 presents the first application (Disassembly Sequence Generator) developed by utilizing the proposed DIM. It focuses on finding all the feasible disassembly sequences from a given EOL product and then locating the optimal one among them. The chapter starts with introducing the role of “Disassembly Sequence Generator” in the overall disassembly planning process and a more specific application level Disassembly Sequence Generator IM is further put forward by extending the proposed DIM in chapter 3. Based on the information provided in the extended DIM, a CAD-API based disassembly sequence generation algorithm is developed to find all the possible disassembly paths of a given EOL product. Lastly, a Linear Programming (LP) model is developed to find the theoretical optimal disassembly sequence among all the possibilities.

Chapter 5 presents the second application (Adaptive Disassembly Planning) developed by utilizing the developed DIM. This application focuses on finding the optimal disassembly sequence, considering economic benefits and product/operation uncertainties. In general, the development of the application follows the same mechanism as described in chapter 4. A specific Adaptive Disassembly Planning IM is developed by reusing and extending the original DIM presented in Chapter 3. The fuzzy logic and Bayesian theorem are combined (for handling uncertain issues) in a developed Disassembly Decision Network (DDN), which are used to adaptively generate optimal disassembly step at each operation stage.

Chapter 6 suggests future works and concludes this dissertation. Extensions of DIM on other disassembly related applications are suggested and contributions of the work are re-emphasized.

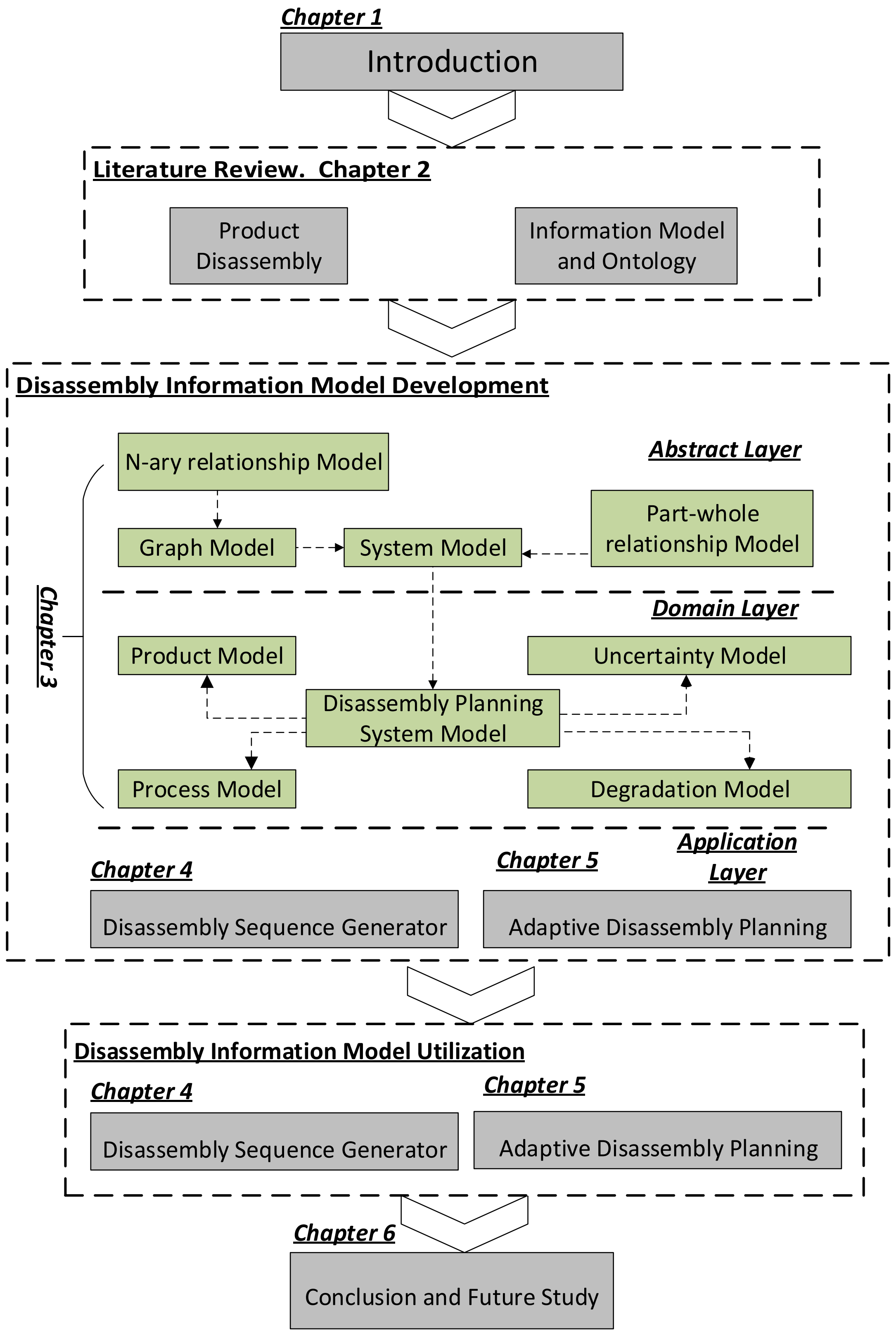

Chapter 1
Introduction
Literature Review. Chapter 2
Product Disassembly
Information Model and Ontology
Disassembly Information Model Development
Chapter 3
N-ary relationship Model
Abstract Layer
Graph Model
System Model
Part-whole relationship Model
Domain Layer
Product Model
Uncertainty Model
Disassembly Planning System Model
Process Model
Degradation Model
Chapter 4
Chapter 5
Application Layer
Disassembly Sequence Generator
Adaptive Disassembly Planning
Disassembly Information Model Utilization
Chapter 4
Chapter 5
Disassembly Sequence Generator
Adaptive Disassembly Planning
Chapter 6
Conclusion and Future Study

Figure 1.2: The Overall Structure of the Dissertation

Chapter 2 BACKGROUND AND LITERATURE REVIEW

In this chapter, a comprehensive review on the scientific background and an establishment of the technical terminologies related to this work have been carried out. Two major topics are reviewed in details: (1) EOL Product disassembly problem and (2) Information Model & Ontology (figure 2.1). The findings and observations from the literature survey has been further analyzed to lighten the potential opportunities (hypothesis) for disassembly planning research.

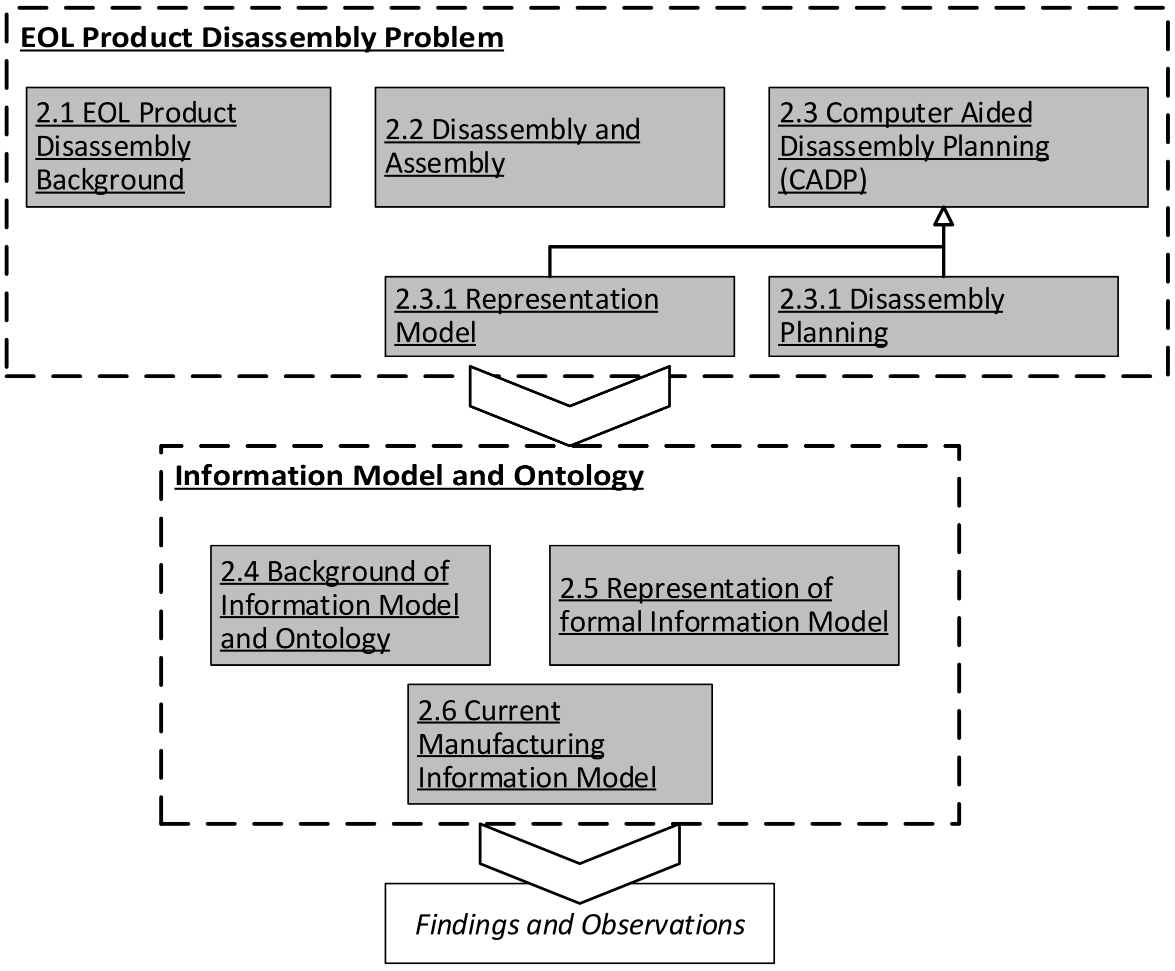

EOL Product Disassembly Problem
2.1 EOL Product Disassembly Background
2.2 Disassembly and Assembly
2.3 Computer Aided Disassembly Planning (CADP)
2.3.1 Representation Model
2.3.1 Disassembly Planning
Information Model and Ontology
2.4 Background of Information Model and Ontology
2.5 Representation of formal Information Model
2.6 Current Manufacturing Information Model
Findings and Observations

Figure 2.1: The Structure of the Literature Review Chapter

2.1 EOL Product Disassembly

In the past decade, the majority of discarded electronics has been destined for landfills and incineration with few economic considerations (Clegg & Williams, 1994). A large amount of potential "residual value" in the EOL product, which could have been recovered through recycling or reuse activity, is usually overlooked. By 2020, the number of discarded computers, televisions, and other electronics containing hazardous as well as valuable materials could reach nearly three billion units (Ilgin & Gupta, 2010). This calls for a systematic management strategy for EOL product to achieve both optimal economic benefits and minimum environmental burdens.

A lot of industry attempts have already been made to address this issue. For examples, at the Reutilization Center in Endicott, New York, IBM has laid out two disassembly lines—a stationary disassembly line for larger computer machines and a conveyor-driven disassembly line for personal and notebook computers (Grenchus, Keene & Nobs, 1997). The process mainly includes customer shipment, receipt and inventory verification, process preparation, disassembly, sorting, and component recovery. Sony has also incorporated the Design for Environment (DFE) principle into its product development process. At the Sony Disassembly Evaluation Workshop in Stuttgart, Germany, products are taken apart to assess the reuse and recycling qualities of electronic parts (Ridder & Scheidt, 1998). The recovery facility can handle a set type of products which include television, compact stereo system, etc. Every step during the disassembly process is clearly documented and evaluated to help improve the future designs.

However, most of the existing EOL product recovering facilities are still following an ad-hoc process when specifying the detailed steps (like disassembly sequences, recovery option selection, etc.) in the product recovery, which makes the whole process economically non-optimized. Also, the existing recovery facilities are operated by big companies like IBM that can only handle a certain type of products specific to those companies; whereas a general independent recovering facility serving a wider range of products and companies is still not available.

2.2 Disassembly and Assembly

Disassembly, as the core step in the EOL product recovery, is defined as "A systematic method for separating a product into its constituent parts, components and subassembly" (Gungor & Gupta, 1999). A common misunderstanding is that the product disassembly process is the reverse of the assembly process. Although one of the major incentives for studying the disassembly process in a systematic way does come from the success of assembly planning, there are still critical differences between the two domains of interest, which should not be overlooked:

1. The assembly process is deterministic in nature, whereas the disassembly process has a lot of uncertainty issues. First of all, the products that come after their end of life services for disassembly purposes are not the same, even though they were the same initially (at the beginning of their product life). An example of such a case can be the same products with different configurations (the user have added one memory card on his PC). Second, the part might be broken or deformed after usage, thus the quality of the part is uncertain. Third, the disassembly operation might not be successful all the time, a damage to the component could have occurred during the disassembly process, possibly due to the harshness of the disassembly process or due to operator error.
2. The objective of the disassembly process is to maximize the profits and/or minimize the environmental impact and thus a complete disassembly is not always the target. Thus, a concept called "disassembly depth" is introduced (Giudice, 2010), which deals with how much effort should be expended in the disassembly of a product, or alternatively, how completely a product should be disassembled. Such a "disassembly depth" has to be determined (and probably adaptively modified) for each individual product. On the other hand, assembly

process follows a fixed assembly plan and targets on optimizing certain performance indicators like throughput, machine utilization, etc.

2.3 Computer Aided Disassembly Planning (CADP)

Researchers are looking for tools and methods for aiding the disassembly planning process, and they advocate the development of a Computer Aided Disassembly Planning (CADP) system. A general structure of CADP can be represented in figure 2.2 below.

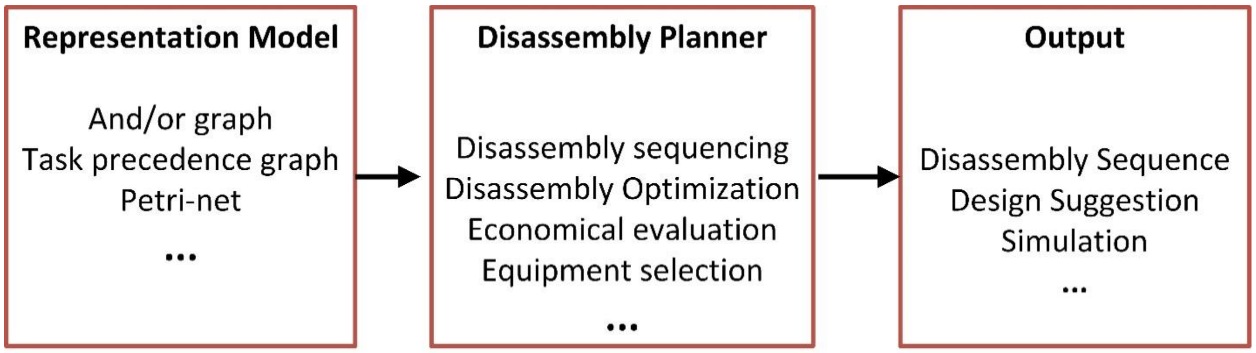


Figure 2.2: The General Structure of a CADP System

As it is evident in figure 2.2, the overall structure of a CADP system can be devided into three layers: (1) Input Layer (Representative Model and database), (2) Computational Layer (Disassembly Planner) and (3) Presentation Layer (Outputs). This dissertation mainly focuses on the work related to the first two layers and notable relevant works are presented in the following sections.

2.3.1 Representation Model

The representation model constitutes the main input for a disassembly planning system and its main objective is to describe the relevant features of an EOL product or a disassembly process. Two main representation models used in this thesis are briefly reviewed here:

(1) AND/OR Graph

An AND/OR graph (Homem de Mello & Sanderson, 1989) is a directed graph G = (N, D), where N stands for nodes that denotes a product, part or subassembly. D stands for hyper arcs which represents the set of feasible disassembly operations. Each node i can have k (k>=0) disassembly choices, forming an OR-relation; an operation disassembles node i into m (m≥2) nodes, m arcs link node i to these m-nodes, and form an AND-relation. Figure 2.3 is a simple example of the AND/OR graph of a product. Arc 1 in the figure represents disassembly operation 1 and the assembly ABCDE can be disassembled into subassembly ABCD and part E (which is not shown in figure 2.3) through the disassembly operation 1. Similarly, operation 3 disassembles subassembly ABCD into subassembly AB and subassembly CD. Each path in the AND/OR graph forms a feasible disassembly sequence. As an example, path 0-1-3 in figure 2.3 is one of the feasible disassembly sequences of product ABCDE (operation 0 is a pseudo operation denoting the initialization of the disassembly process).

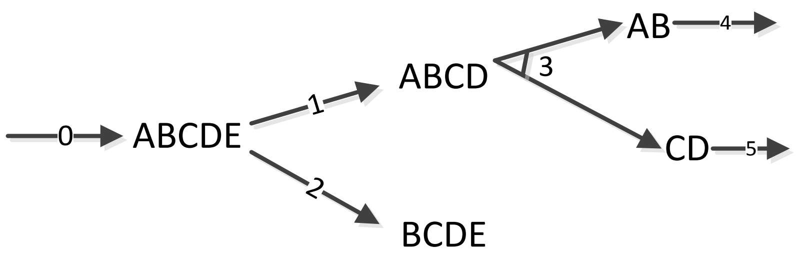


Figure 2.3: An Example of the AND/OR Graph

(2) Task precedence graph

Instead of representing nodes as parts and sub-assemblies, nodes represent disassembly operations in the task precedence graph. Two disassembly operations are represented by two nodes connected by a directed arc signifying one operation proceeded by the other. If the AND/OR graph in figure 2.3 is translated into a task precedence graph, it will look like that which is shown in figure 2.4 below.

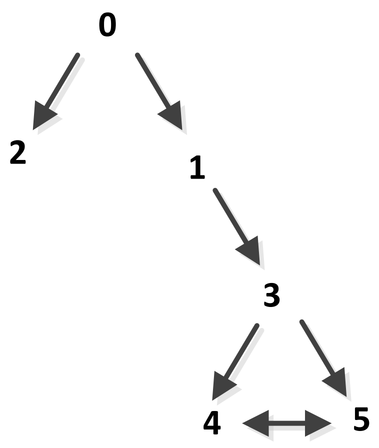


Figure 2.4: An Example of the Task Precedence Graph

Please note that the Operation 0 has been considered as a pseudo operation and it is the initialization of the disassembly process. After initialization, either operation 1 or operation 2 can be executed. The doubly directed arc means either one operation can be done before or after the other one, e.g. operation 4 can be done after operation 5 and vice versa. Though the task precedence graph is a derivative of an AND/OR graph, it has the advantage in that the sequence-related information is easily observable in the task precedence graph (Any goal node could be arrived at following more than one path from a given starting node in the task precedence graph), whereas such information is implicit in the AND/OR graph. For example, it is not clear from the AND/OR graph that operation 4 can be done after operation 5 (figure 2.3) (Zhu, Sarigecili & Roy, 2013).

Besides the above representation model, other similar modeling derivatives have been proposed, which includes disassembly petri net (Zussman, MengChu Zhou, & Caudill, 1998), connection diagram (Lambert, A. J. D. (Fred) and Surendra M. Gupta, 2005), state diagram (Lambert, A. J. D. (Fred) and Surendra M. Gupta, 2005), etc. A good description of these models can be found in (Ghandi & Masehian, 2015).

2.3.2 Disassembly Planning

Based on the different representation model, different planning approaches and methods have been proposed for the disassembly planning problem and most of them fit into the following categories:

(1) Graph-based approach

Graphs usually represent the structure of a system, process, product, organization, etc. They can be considered as an abstraction of the reality. Graph theory has been used as a powerful tool to solve the problems of disassembly planning. It has helped in representing the planning process by providing tools like connection diagrams and AND/OR graphs. The characteristics and functions of a disassembly system are explicitly expressed in the graph and different searching algorithms are further applied to find all the feasible disassembly sequences according to the topological, geometrical and technical constraints. Different strategies are further applied to locate the optimal sequence with consideration of the plan effectiveness and cost-effectiveness. Several outstanding graph-based approaches are briefly discussed below.

Penev et al. (Penev & de RON, 2002) used AND/OR graph theory and methods of dynamic programming for the generation and evaluation of the feasible disassembly plans. A new economic model is introduced to determine the optimal level of disassembly. Zhang et al. (Zhang & Kuo, 1996) developed a graph based heuristic approach for the generation of disassembly sequences from CAD system directly. They proposed a component fastener graph to analyze the product assembly relationship. A search for a set of cut-vertex and decomposition of the EOL product into several subassemblies is further applied on the graph to simplify the disassembly analysis process. Murayma et al. (Murayama, Oba, Abe & Yamamichi, 2001) described the disassembly sequence generation using the idea of information entropy and heuristics to replace components at maintenance stages. The advantage of this method is primarily in the reduction of searching time and searching places for disassembly sequences. The author also developed a software tool integrated with a CAD system and carried out an experiment for an electric drill using the tool. A graph-based information modelling system to represent the process for disassembly and recycle planning of consumer products was proposed by Kanai et al. (Kanai, Sasaki & Kishinami, 1999). Four kinds of graph have been presented: (1) a configuration graph of sub-assemblies or fragments; (2) a connection graph between parts and materials; (3) a process graph of disassembly, shredding, and sorting activities; (4) a retrieval condition graph. Rules and procedures for transforming the models of these activities are uniformly formulated. A vacuum cleaner is used as an example to demonstrate the proposed graph-based method.

(2) Petri net-based approach

Besides the traditional graph-based disassembly analysis approach, Petri-Net (PN), as a graphical and mathematical tool, provides a uniform environment for modelling and analyzing both static and dynamic discrete events. They provide a very promising method for disassembly sequence generation.

Zussman et al. (Zussman, MengChu & Caudill, 1998) proposed a complete and mathematically sound Disassembly Petri Net (DPN) approach to model the disassembly processes. In their work, the detailed construction and advantages of the proposed DPN have been discussed and a DPN based searching algorithm has been proposed for the generation of the disassembly plan. They further extended this work (Zussman & Meng Chu Zhou, 2000) and proposed a design and implementation system for an adaptive process planner for disassembly processes. The system also incorporates the uncertainty issue caused by the different product conditions.

Moore et al. (Moore, Gungor & Gupta, 1998) developed an algorithm for automatically generating a DPN from a disassembly precedence matrix. The DPN representing the specific precedence relationships among parts can be derived from a CAD representation of the product. A Reduced Reachability Tree algorithm has been further proposed to identify the near-optimal disassembly process plan from using the DPN.

(3) AI based approach

Many attempts have been made using Al techniques (Genetic algorithms, ant colony methods, fuzzy logic, neural networks, etc.) in the disassembly sequence optimization. The objective is to reduce this time by searching the best disassembly sequences without analyzing all the possible alternatives. Several examples are discussed as below:

An example of the use of fuzzy logic in disassembly planning is proposed by Chevron et al. (Chevron, Binder, Horacek & Perret, 1997). The main goal is to find the disassembly sequence requiring the minimum completion time, taking into account the fuzzy model of the processes and the constraints in available tool, destruction modes, etc. The problem of the generation of disassembly sequences is approached as a travelling salesman problem (the traveler is the product and the cities are the operations with their processing times). A modified branch-and-bound method is used with an objective function evaluated according to fuzzy parameters.

Hsin Hao et al. (Hsin-Hao, Wang & Johnson, 2000) proposed a Neural Networks approach to the planning of disassembly problem. The generation of sequences is again viewed as a variant of the traveling salesman problem: to find the sequence of components to be disassembled (cities) having the greatest profit (the shortest distance). This problem is approached using a Hopfield Neural

Network. As input, an N by N matrix of neurons is used: the rows of the matrix indicate the disassembly operations to be scheduled, and the columns the disassembly sequences.

Lambert (A. J. D. Lambert, 1997) proposed a Linear Programming (LP) model to the disassembly planning problems. The LP model tries to find the optimal disassembly sequence based on maximizing the total value of the retrieved parts/subassembly and minimizing the total disassembly operation cost associated with them.

Table 2.1: Summary on the Reviewed Disassembly Planning Work

Author	***Representation Model***	***Information Involved***
Penev et al. 2002	AND/OR graph	Product, Process
Zhang and Kuo, 1996	Component-Fastener Graph	Fastener, Product
Murayma et al. 2001	Information entropy embedded product graph	Product
Kanai et al.	Configuration graph Connection graph Process graph Retrieval condition graph	Process Condition Information Product
Zussman et al. 1998	Disassembly Petri Net	Process, Product
Moore et al. 1998	Disassembly Precedence Matrix Disassembly Petri Net	Process, Product
Chevron et al. 1997	Fuzzy Logic based Process and Equipment Model	Process, Product, Uncertainty
Hsin Hao et al. 2000	Disassembly Neural Network	Product, Process
Lambert, 1997	AND/OR graph	Process

A summary on the disassembly planning methods is presented in table 2.1. One important observation can be identified here: although different researchers proposed different representation models, the involved information (product, process, etc.) shared similarities among different methods. The reuse of these concepts has not been explored, which could have made the development processes of the CADP applications less time consuming.

2.4 Background of Information Model and Ontology

The Information Model, sometimes called ontology, is the consensual modelling of concepts and relationship in a domain of interest. In this dissertation, we use the term IM and ontology interchangeably. The word “Ontology” can be traced back to the 4th Century BC and is originally a philosophical discipline concerned with the question of what exists and what is the essence of things (Zuniga, 2001). Over the last decades, it has been adopted by computer scientists, firstly in the field of Artificial Intelligence (AI) and more recently in other engineering areas like biology, chemistry, medicine, etc. Within this community (engineering community), the term is used in a narrower sense than that in the context of philosophy. It emphasizes on a formal representation of contextual information, which contains precise definitions of certain entities in terms of their properties and their relations to other entities. Such definitions are usually given in the form of axioms formulated in a logic-based language, which can facilitate the automated knowledge reasoning process (Kutz & Garbacz, 2014).

2.5 Modeling Elements for the Formal Information Model

Information Model, as a conceptual model, can be constructed with different modeling techniques and be implemented in various kinds of languages (Uschold & Gruninger, 1996). Over the years, researchers have explored different modelling paradigms such as description logic (Mann, 2003), database modeling techniques (Bera, Krasnoperova & Wand, 2010), Semantic Web approach (Memon, Ortiz-Arroyo & Larsen, 2005), etc. Despite the diversity among different approaches, all of them have common modeling elements. In particular, most languages provide constructs for classes, instance, relations, and attributes, although they may be named differently in the respective implementations.

Class: A class represents a set or a category of things that have some properties or attributes in common and they are differentiated from others by kind, type, or quality. It sometimes can also be denoted as concept or frame depending on the different modelling paradigms. An example of a class could be **Product**, **Disassembly Process**, **Constraint**, etc. (We will use the bold Calibri font, with the capitalized first letter, in this dissertation to represent a class).

Instance: Entities that belong to a particular class are said to be instances or members of that class; for example, *steel* and *plastics* are instances of the **Material** class. (We will use the italicized Calibri font, with the lowercased first letter, in this dissertation to represent an instance of a class)

Relations: Relation describes the interrelation between classes and it can also be denoted as properties, roles, slots, or associations in other modelling paradigms. Most modelling languages support representing only relations among two classes and is by default directional, which means that it points from a particular domain class to a designated range class. As an example, consider the relation ***hasComponent***, which refers from a **Product** (its domain) to a **Component** (its range). We will use the bold italicized Calibri font, with the lowercased first letter, in this dissertation to represent a relation. A special relation called inheritance relation is commonly supported in various IM modeling paradigm, which is used to hierarchically organize the classes by specifying parenthood relations. As an example, a **Screw Connection** class is inherited by **Connection** class and it is a specialization or a subclass of the **Connection** class (every instance of **Screw Connection** is also an instance of **Connection** class).

Attribute: Attributes represent features, characteristics, or parameters of classes and an attribute is identified by a name and can take one or several values, which are usually restricted to a specific datatype such as Boolean, string, integer, etc. We use the underlined Calibri font, with the lowercased first letter, in this dissertation to represent an attribute of a class. As an example, manufacturingCost is an attribute of class **Component** and it can take values of datatype double.

2.6 Current Information Model in the Domain of Manufacturing

Over the years, researchers have contributed to the development of IM or ontology in the domain of manufacturing, with different focusing aspects. Some notable work is reviewed below.

Leimagnan et al. (Lemaignan, Siadat, Dantan & Semenenko, 2006) developed the Manufacturing Semantic Ontology (MASON) to formally capture the concepts related to the manufacturing industry. The semantics related to entity, resources and operation were captured in formal logic using web ontology language (OWL). Two applications about automatic cost estimation and the semantic-aware multi-agent system for manufacturing were discussed to demonstrate the usefulness of the proposed MASON ontology.

Xiaomeng (Chang, 2008) selected the field of Design for Manufacturing (DFM) for his PhD study and three primary aspects are investigated. First, a generalized DFM ontology is proposed and developed, which fulfills the mathematical and logical constraints needed in the domain of DFM. Second, the means to guide users to the proper information and integrate heterogeneous data resources is investigated. Third, a decision support tool is developed to help designers consider the design problem in a systematic way based on the developed DFM ontology.

Pavan (Kumar, 2008) developed an ontology called the Design Activity Ontology (DAO) to explicitly represent the design activity that can cover phases of the design process from conceptual phase through detail design phase. The ontology provides a formalized and structured vocabulary of design activities for the exchange of design process models and it further enables design processes to be modeled, analyzed and optimized in a consistent way.

Kim et al. (Kim, Manley & Yang, 2006) proposed a collaborative assembly design framework that offers a shared conceptualization of assembly modeling, and an Assembly Design Ontology (AsD) is developed to capture the joining intents of a product. AsD is claimed to serve as a formal, explicit specification of assembly design so that it makes the assembly knowledge both machine-interpretable and sharable.

Some industrial efforts have also been devoted to the development of the manufacturing related Information Model. A notable development in this field is led by NIST. One of their work is the NIST's Core Product Model (CPM), which a unified modeling language (UML) based model intended to capture the full range of engineering information commonly shared in product development (Foufou, 2005). CPM focuses on modeling the general, common and generic product information and excludes the information which is domain specific. NIST further developed another information model called "Open Assembly Model" (OAM) (Baysal, 2004) which extends CPM. Along with the structural information, it represents the function, form, and behavior of the assembly, and defines a system level conceptual model.

Recently, NIST also proposed a disassembly information model (Feng & Kramer, 2013) and to the author's knowledge, this is the first attempt to develop disassembly related information model. The developed model highlights the information content used for disassembly sequence representation, feature modeling, equipment modeling, and inspection process modeling. However, the NIST disassembly information model remains in the conceptual stage and the implementation of the model has not been fully achieved. Also, the handling of reusability/usability tradeoff issue and the uncertainty issue is not discussed.

2.7 Summary: Observation and Findings

An in-depth review of the disassembly planning problem and the Information Model has been carried out in this chapter. Some observations and findings are summarized as follows:

Information Model provides a shared knowledge basis for a specific domain of interest and it is necessary for any decision making purposes. Even though some work has been done in developing manufacturing related Information Model, not sufficient attentions were paid towards the issues related to the EOL product disassembly in a comprehensive manner. Even for the NIST disassembly Information Model, certain issues like the reusability/usability trade off and the EOL product uncertainty have not been well addressed.

In the disassembly research area, different representation models like AND/OR graph, Component-Fastener Graph, Information entropy embedded product graph, etc. have been proposed. A finding from the literature survey is that the involved information in different proposed disassembly planning methods shares certain commonalties (product, process, uncertainty, etc.), which should be generalized for better serving the disassembly research community.

Chapter 3 DISASSEMBLY INFORMATION MODEL

In this chapter, the development and implementation of the proposed Disassembly Information Model (DIM) are presented in detail. We start with the discussion and analysis of the information requirements in the domain of disassembly planning, which puts forward a high level informal domain conceptualization (section 3-1). Next, a layered DIM modelling methodology is presented, with the intention to find a balance between IM reusability and usability (section 3-2). The detailed DIM model is introduced afterwards in section 3-3, using UML class diagram as a graphical notation. Lastly, the formal DIM implementation in OWL is presented in detail in section 3-4.

3.1 Information Requirement for Disassembly Planning

The information required for the EOL product disassembly planning can be broken down into four categories: product related, process related, uncertainty related and component degradation related (figure 3.1). The informal description of each category is presented below:

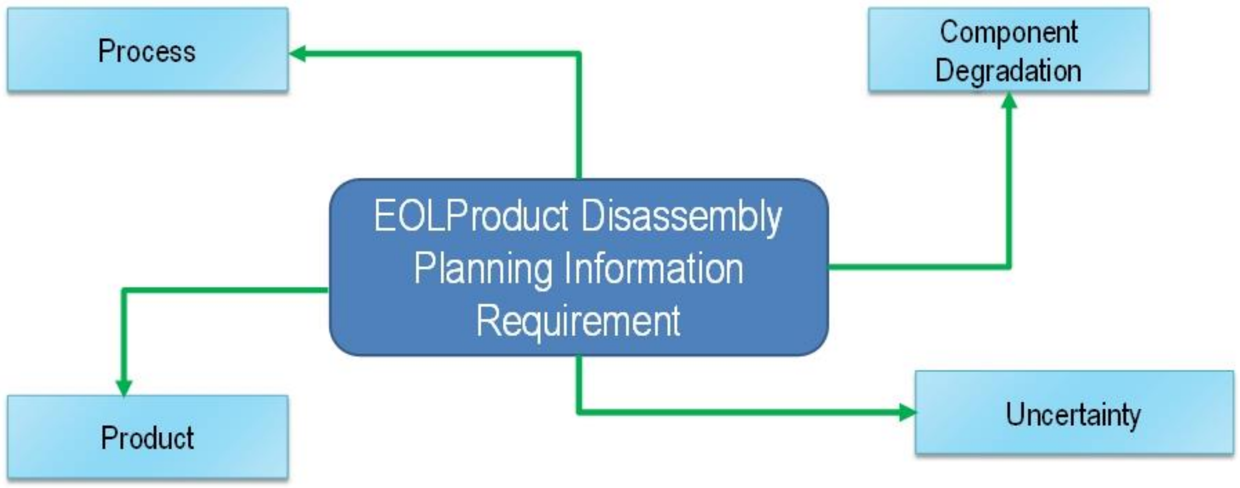


Figure 3.1: High Level Information Requirement for the EOL Product Disassembly Planning

3.1.1 Product Aspect Information

The product related information describes the characteristics of the EOL product which needs to be disassembled. Relevant concepts or terminologies in this domain include product, component and liaison and they will be informally described below to convey a fundamental domain understanding before the formal product aspect Information Model is presented.

Product and Component

In general, a product is an artifact or substance that is manufactured for sale. In any disassembly process, the product represents the input to the disassembly process and it may consist of a number of discrete parts, which are called components. A component is a material entity that can be separated from a product through disassembly processes, without altering the component's intrinsic property (like mass, density, etc.). Furthermore, a component cannot be further separated via non-destructive detaching processes.

In the domain of disassembly planning, further specification of the component according to their characteristics is critical. From a higher level, components can be classified according to different aspects like material composition (homogeneous or composite), functional type (connecting function or non-connecting function) and component complexity (atomic component or complex component). In detail, the following types of component are highlighted:

- Homogeneous Component: is a component consists of only homogeneous materials. Frame and cover are the typical examples of the homogeneous component.
- Composite Component: is a component consists of different non homogeneous materials linked in an irreversible way, such as a sandwich structure. The laminated glass, which is constructed by combining two panes of glass fused together with a middle layer of Polyvinyl Butylenes Film (PVB) acting as a bonding agent, is an example of composite component,
- Connecting Component: is a component whose primary function is to connect other components. Different fasteners fall under this category.

- Complex Component: A complex component is a cluster that consists of a set of components, which cannot be separated from the whole without damaging certain component permanently. Examples of such component can be printed circuit board and electrical cables.

Liaison

Components are physically linked by liaison, which restricts the freedom of motion of the components involved. The liaison concept can be classified into two main types to reflect the different properties of the liaison. The main liaison types are:

- Component contact: Such liaison represents the relationship between components where the involved components are connected with each other without any application of external forces. We call this type of connection “component contact” and it is formed through connections between component’s geometric entities like a vertex, an edge or a surface. Examples of such case could be a cube resting on a panel (surface contact).
- Component connection: Such liaison represents the relationship between components where a connection is established through a certain connecting component. Example of such case could be a blender housing connected with a base panel by a set of screws (connecting component).

From the discussion above, two Information Modeling requirements related to the product domain aspect can be identified. The first one is the modeling of product hierarchy: a product is composed of different components which are hierarchically organized by aggregating them into subassemblies. Thus, the part-whole relationship needs to be modeled in the product domain Information Model. Second modeling requirement relates to the topological arrangements of components (or say product structure), which are realized by different component liaisons.

Information related to how components are connected to each other for achieving the final product should be supplied. If the liaison belongs to the type of component connection, at least three entities are then involved: two components are connected through one connecting component. Thus, modeling of n-ary relationship (n>2), which involves more than two entities, should be supported.

Basic product aspect information requirements can be summarized in table 3.1 below:

Table 3.1: Product Aspect Information Requirement

Basic Terminology	Product
	Component • Homogeneous component • Composite component • Connecting component • Complex component
	Liaison • Component Contact • Component Connection
Modeling Requirement	Product Hierarchy • Part-whole relationship
	Product Topology (Component Liaison) • N-ary relationship

3.1.2 Process Aspect Information

The disassembly process accomplishes the basic transformations of the product's physical states and it can be divided into three different levels as follows:

- Task Level: Task represents the most abstract type of disassembly process, which only specifies the target component to be disassembled. An example of a disassembly task could be "detaching blender housing component" or "disassembling the screw from the PC motherboard". A sequential aggregation of disassembly tasks will provide a high level disassembly plan.

- Operation Level: An operation represents the detailed process steps necessary to achieve a certain task. The operation may not only include disconnection process, they may also include the movement operations necessary to transfer the subassemblies to a different location and other supplementary operations such as cleaning, fixturing, tool exchanging, product reorienting (to guarantee access or stability), and testing.
- Action Level: An action represents the specific atomic process steps required to achieve a certain operation. An important characteristic of an action is that it is performed without the goal to directly change the object state (Hamidullah, Bohez & Irfan, 2006). It means that an action alone should not be sufficient to change any part attributes or disestablish of liaisons. As an example, a "movement" operation involves possibly two actions: motion action and grip action. However, neither the motion action, nor the grip action alone changes the state of the object (A motion action will not make any difference on the component unless it is combined with a grip action).

Along with the detailed process classification, another important process related requirement is the ability to represent all the feasible disassembly process sequences explicitly. In other words, this requires the development of an Information Model that can mimic the traditional graph based disassembly process representation, such as the state change graph. This requirement utilizes again n-ary relationship because a state change normally involves three objects: pre-state, goal-state and process step.

Basic process aspect information requirements thus can be summarized in table 3.2 below:

Table 3.2: Process Aspect Information Requirement

Basic Terminology	Process • Task • Operation • Action
Modeling Requirement	Feasible Process Sequences • N-ary relationship Process Hierarchy • Part-whole relationship

3.1.3 Uncertainty related Information

As mentioned in the previous sections, unlike the assembly process, the disassembly process has various inherent uncertainty issues. Thus, extra information is needed for such uncertainty handling. Two types of uncertainty are considered in this dissertation: (1) Component/assembly function uncertainty and (2) Operation uncertainty.

- Component/assembly functional uncertainty: each component or assembly might associate with a primary function, which contributes to the product overall function. Such function may not be working when the EOL product becomes obsolete. Such functional/non-functional information is critical in the disassembly planning process and can only be revealed gradually during the disassembly process.
- Operational uncertainty: during the disassembly process, certain operations such as unscrewing might not succeed due to the poor physical conditions of the component such as deformation or corrosion. In such cases, extra special operations may be required to handle the situation and it will incur a higher cost. Since this information is also unknown at the beginning of the disassembly process, it is called operational uncertainty.

Both cases will be handled using the Bayesian Network (BN), which consists of a Directed Acyclic Graph (DAG) and a set of local statistical distributions (Kwaan, 1994). The detailed procedures for disassembly uncertainty handling using BN will be presented in detail in the chapter 5. Here we only summarize the two important information elements necessary for the Bayesian theorem based uncertainty handling: (1) the component/assembly influence dependency and (2) the conditional probability table.

Component/assembly Influence dependency: is a directed acyclic graph (DAG) representing the function dependency among components/assemblies. Figure 3.2 is a simple DAG example describing that the function of the "Fan Assembly" is conditionally dependent on the function of the "Motor" and function of the "Rotor Shaft". Such information demonstrates the function failure propagation in the EOL product and is critical in the adaptive disassembly planning.

Conditional Probability Table (CPT): consists of a set of discrete (not independent) random variables to demonstrate the marginal probability of a single variable with respect to the others. A simple CPT applied to the example in figure 3.2 is shown in figure 3.3. It says that the probability of the "Fan Assembly" to be functional is conditionally dependent on two other variables: the probability of the "Motor" to be functional and the probability of the "Rotor Shaft" to be functional. As an example, when the motor is functional and the rotor shaft is not functional, the probability of the fan assembly to be functional is 0.

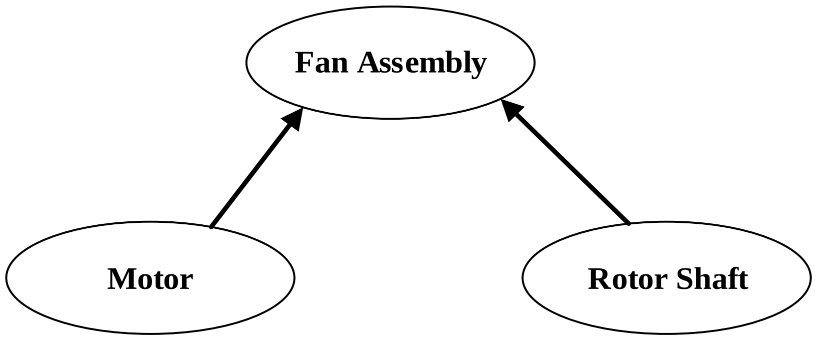


Figure 3.2: A Simple Example of the DAG Network

	Motor	Function		Not Function	
	Rotor Shaft	Function	Not Function	Function	Not Function
Fan Assembly	Function	0.8	0	0	0
	Not Function	0.2	1	1	1

Figure 3.3: CPT of the Fan Assembly

In summary, uncertainty related information requirement can be summarized in table 3.3 below:

Table 3.3: Uncertainty Related Information Requirement

Modeling Requirement	Bayesian Networks • Conditional Probability Table • Component/assembly Influence dependency

3.1.4 Degradation related Information

Component/assembly degradation is also a critical issue in the planning of disassembly. Degradation is a gradual change in the properties (like tensile strength, color, shape, etc.) of the component, which usually does not affect the overall function of a component until it reaches a critical point. However, degradation does affect the economic quantification of EOL product or component. For example, although some subassembly might work fine (functional) after the function testing, the associated reuse value still could be lower than the expected average reuse value (the subassembly is close to failure) or higher than the expected average reuse value (the subassembly still has a long remaining useful life time). Information regarding to such remaining useful life estimation should be supplied for the disassembly planning process.

The remaining useful life time estimation is a challenging research problem and in this thesis we use the fuzzy logic based approach to quantify the component/subassembly reuse value through a set of fuzzy linguistic variables and a set of heuristic rules. The detail of reuse value estimation using fuzzy logic will be presented in detail in the following chapter (chapter 5). Here we only summarize the important information elements necessary for carrying out the fuzzy logic based reuse value estimation.

- Age: age represents the service time of a component or a product. This information could be different among different components in the one product (component replacement during the maintenance). Usually, high age indicates a lower reuse value.
- Condition parameter: age is an indicator variable for estimating the remaining component useful lifetime. However, it is assumed that the component or subassembly is servicing under certain controlled operational conditions. If the user is abusively using a certain product or a product is operating under severe external environments, the age of the component/product alone can no longer properly indicate the remaining useful life time. Certain condition parameters (like operation noise, corrosion, etc.) should be included for the estimation.
- Market demand: reuse value is also dependent on the market demand. A higher demand normally will increase the average reuse value and a lower demand will decrease the reuse value despite of the conditions of the product/component.

The above input variables will be modeled as linguistic variables, which is suitable for fuzzy reasoning. Extra informational elements related to the linguistic variable are thus necessary, which include membership function and fuzzy term definition. On top of that, the support of heuristic based fuzzy rules should be provided in the developed DIM for the fuzzy reasoning process.

Basic uncertainty related information requirements can be summarized in table 3.4 below:

Table 3.4: Degradation Related Information Requirement

Modeling Requirement	Fuzzy Logic • Linguistic variable ○ Age, Condition Variable, Market Demand ○ Membership function ○ Fuzzy term • Fuzzy Rules

3.2 Layered DIM Modelling Methodology

From the analysis carried out in the previous section, DIM should be comprised of the information related to the aspects of product, process, uncertainty and degradation and the modelling of which involves certain information modeling patterns like n-ary relationship, part-whole relationship, etc. Also, DIM should achieve certain balances between IM usability and reusability. Thus, a layered modelling methodology has been proposed, in which DIM has been subdivided by means of layers (Figure 3.4), with the intention to separate general knowledge into different level of abstractions. Also, a "minimal ontological commitment" (Gruber, 1995) guideline is followed, which means each layer holds only concepts/relationships and axioms that are essential for the function of the current layer. Information that is not essential for the layer's purpose are sourced out to lower layers. Details of each layer are presented as follows:

- Abstract Layer: The Information Models in the abstract layer hold the fundamental modeling concepts, which are independent of a particular problem or domain and can therefore be universally applied. They describe the design guidelines (design pattern) for the construction of the other sub models in the DIM. Models like n-ary relationship, part-whole relationship, graph model and system model belong to this layer.

- Domain Layer: The Information Models in the domain layer capture the knowledge related to a domain of expertise, such as disassembly planning in our case, and they generally don't target on solving a specific problem or task, but rather providing a domain knowledge foundation for a range of different applications. Thus, the Information Model residing on this layer is more specific than those in the abstract layer, but less specific than those in the lower layer (application layer). The majority of the required disassembly domain information discussed in section 3-1 (product, process, etc.) are implemented in the models in this layer.
- Application Layer: represents the most specific Information Model which is directly usable for a certain disassembly planning application. This thesis focuses on two disassembly planning applications: (1) Disassembly Sequence Generator and (2) Adaptive Disassembly Planning and they will be discussed in detail in chapter 4 and chapter 5 accordingly.

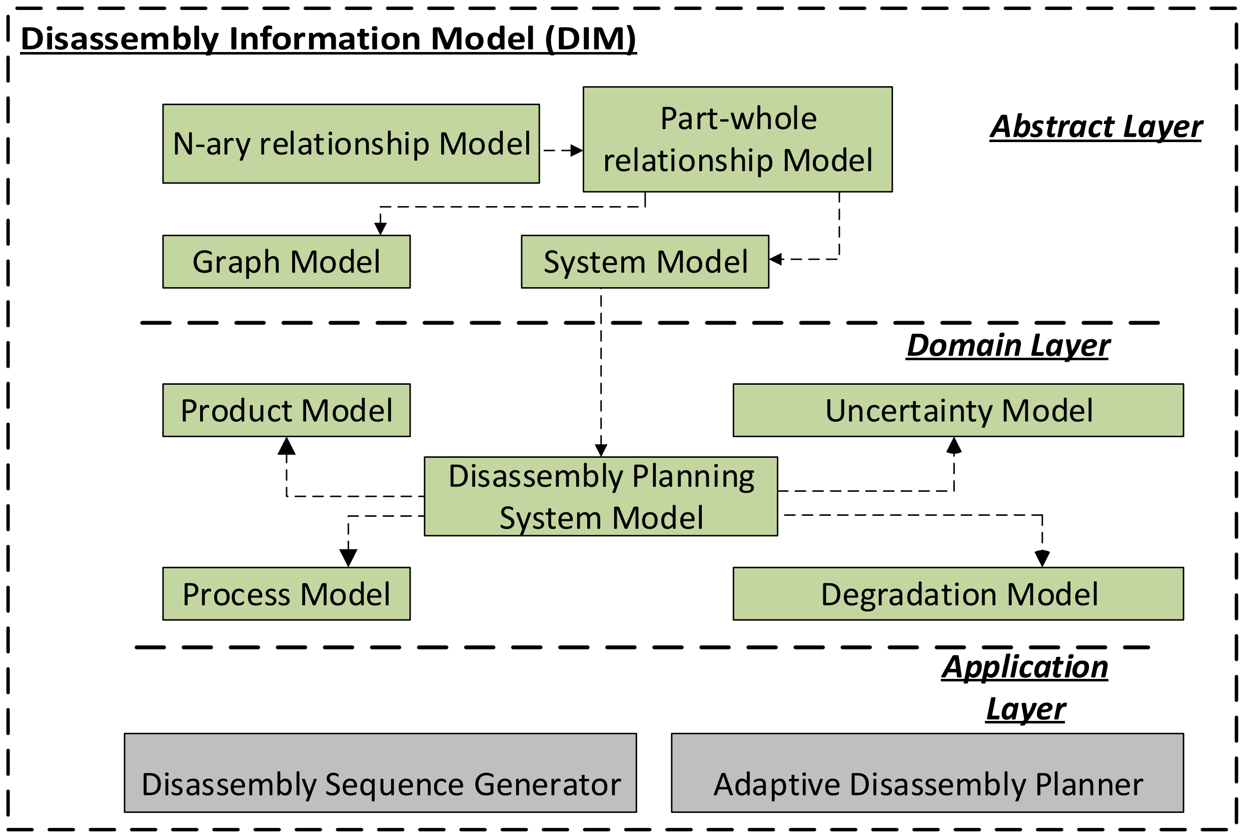

Disassembly Information Model (DIM)
Abstract Layer
N-ary relationship Model
Part-whole relationship Model
Graph Model
System Model
Domain Layer
Product Model
Uncertainty Model
Disassembly Planning System Model
Process Model
Degradation Model
Application Layer
Disassembly Sequence Generator
Adaptive Disassembly Planner

Figure 3.4: The Overall Structure of DIM

Such a layered DIM development methodology takes the IM reusability-usability trade-off problem into account. The abstract or general knowledge is modeled in the sub models located on the top layer of the DIM. They provide various design patterns which can be reused in various application contexts and normally are not directly usable in any particular application due to the high level of abstraction. On the other hand, knowledge in the models residing on the lower layer is ready to be used, but is usually application specific and thus is hardly to be transferred to other applications. Information Models in each layer of the DIM contain knowledge with certain degrees of reusability and usability and the usability of the knowledge normally increases with descending reusability when navigating from the top to the bottom layers of DIM.

In the following sections, DIM sub models in the abstract and domain layer will be presented in detail, whereas the sub models in the application layer will be introduced in chapter 4 and chapter 5.

3.3 Formal Disassembly Information Model

In this thesis, the UML class diagram, which has the full modelling capabilities to represent the major elements (class, relations, etc.) of an Information Model, has been adopted as a graphical representation of the Disassembly Information Model. The UML class diagram notations are summarized in figure 3.5 and they will be applied throughout this thesis.

Rectangular box with bold text represents a certain class and the instance of the class is denoted as italicized regular text in a rectangular box. A class can have some attributes (sometimes called data property), which can hold certain datatype. This is represented as a straight line connection between a class and a data type (represented as rectangular boxes with dashed boundary lines). The name of the attribute (data property) is annotated on top of the connection line as regular text with a lowercase first letter.

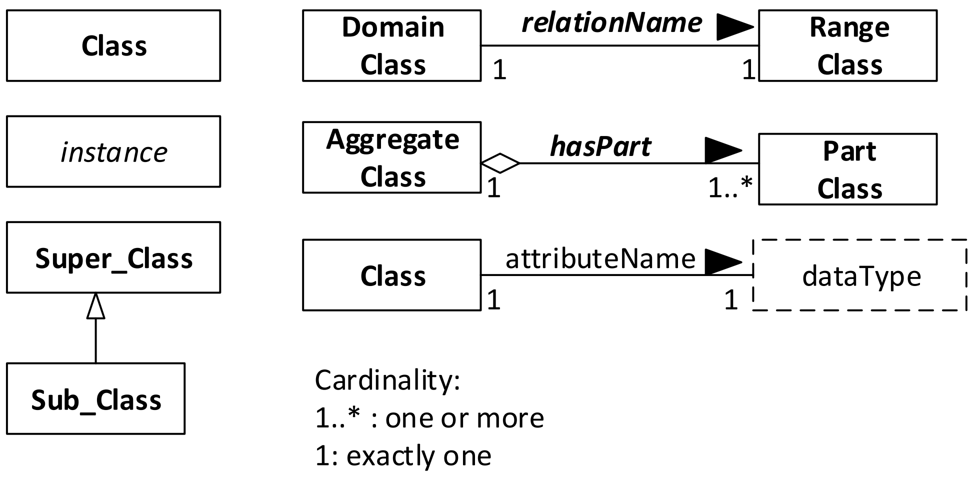


Figure 3.5: The Notations Used in Creating the UML Class Diagram

The inheritance relationship between classes is depicted through a solid line with a hollowed arrowhead pointing from the subclass to the superclass. Binary relationships (sometimes called object property) can exist between classes, which is denoted as a straight line connection between classes. The name of the relationships is presented on top of the connection and cardinality constraints (depicted by numbers placed close to classes of the respective relation) can be added if necessary. Lastly, aggregation, as a special type of relationship, is important in this work, which will be discussed in detail in the part-whole relationship sub model section. The annotation for such relationship is a straight line connection with a hollowed diamond head pointing at the aggregated class.

3.3.1 Abstract Layer Models

This section presents the Information Models residing on the abstract layer in detail.

(1) N-ary Relationship Model

N-ary relationship model presents the most fundamental modeling elements (concept and relationship) in an Information Model. On top of that, we extend the traditional binary relationship into the N-ary relationship, which can represent certain relations existing among more than two objects. Figure 3.6 shows the N-ary relationship Information Model: everything is considered as either an **Object** or a **Relationship**. A **Relationship** class involves two or more **Objects** and could have certain attribute (relationAttribute) with different possible datatypes. In some scenarios, directed N-ary relationship is necessary, which describes an N-ary relationship existing among some **Objects** where at least one **Object** is distinguished as the origin of the relationship. The **DirectedRelationship** class is thus modeled as an extension of the **Relationship** class and two new object properties (***hasOrigin*** and ***hasTarget***) related to the **DirectedRelationship** class are introduced to denote the direction among the objects involved in a relationship.

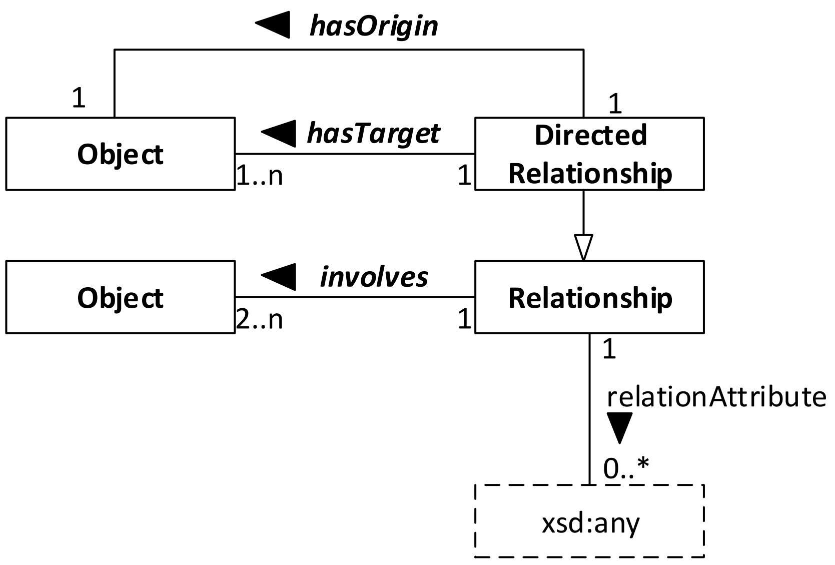


Figure 3.6: N-ary Relationship Information Model

As indicated before, the Information Models in the abstract layer provide a generic information design pattern irrespective of its use in any particular application domains. Here, a simple application example of representing an array ([a, b, b, c]) is shown in figure 3.7 below:

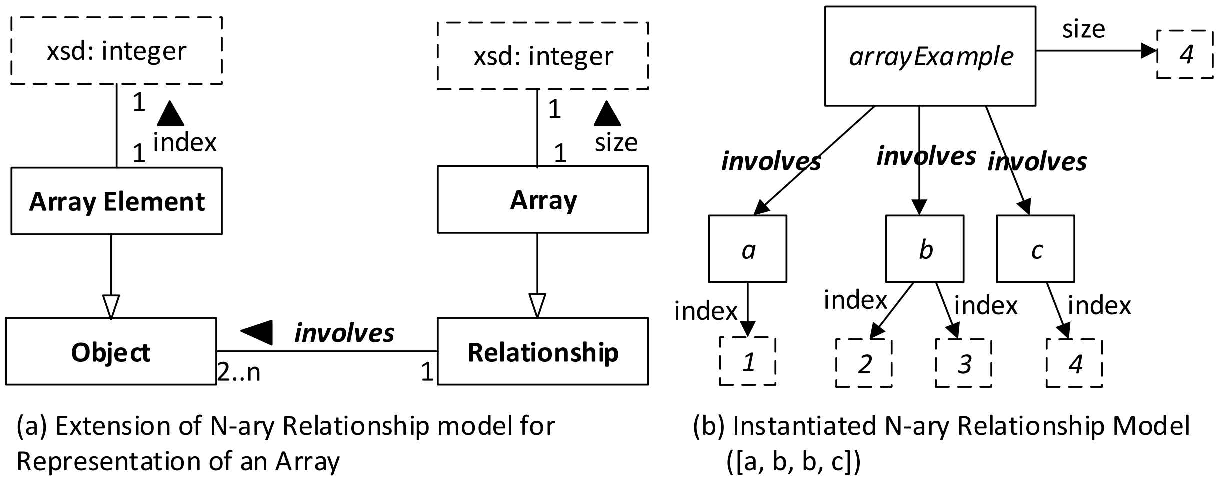


Figure 3.7: An N-ary Relationship Example

An array involves several elements (in our case, English letters) in a sequential way. Thus, an **Array** can be modelled as an extension of the **Relationship** class, which involves several **Array Elements** (extension of the **Object** class). Each of the elements in an array has an index indicating its position in the array and such information is modeled by the index data property, which associates each **Array Elements** with an integer. The size information of an array can be considered as a certain array attribute and thus data property size is modeled and it is an extension of the relationAttribute data property. Figure 3.7 (b) shows how to represent array [a, b, b, c]: *arrayExample* is being modeled as an instance of the **Array** class and *a*, *b* and *c* are instances of the **Array Elements** class, which are being involved in the *arrayExample* relationship. The information regarding the array size and the index of the array element is also shown accordingly.

(2) *Part-Whole Model*

The Part-Whole model represents the parthood relations among **Objects**, which is a common scenario in the domain of disassembly. As examples, parthood relations can exist between product and subassembly, between subassembly and component, between process and task and so on. The Part-Whole model (Figure 3.8) is developed to represent a reusable design pattern for such purpose. Two new classes are introduced: The **Whole** class represents the **Object** which will aggregate other **Objects,** whereas the **Part** class represents the **Object** which is a part of the **Whole** class. An instance of the **Whole** class must relate to some (more than one) instances of the **Part** class through the ***hasPart*** object property.

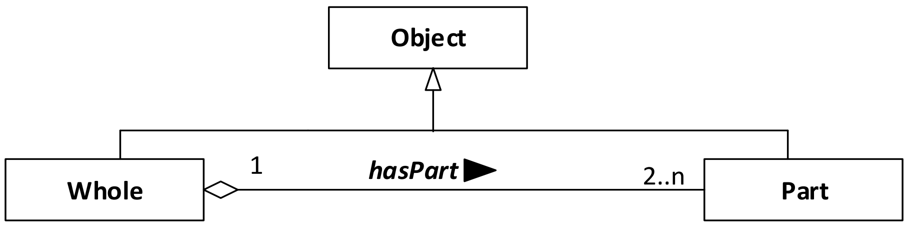


Figure 3.8: Part-Whole Model

(3) Graph Model

Graph is widely used for the representation of disassembly process or the structure of the EOL product, thus the third Information Model in the abstract layer is the graph Information Model, which is extended based on the N-ary relationship model and part-whole model. Figure 3.9 presents the overall structure of the Graph Information Model.

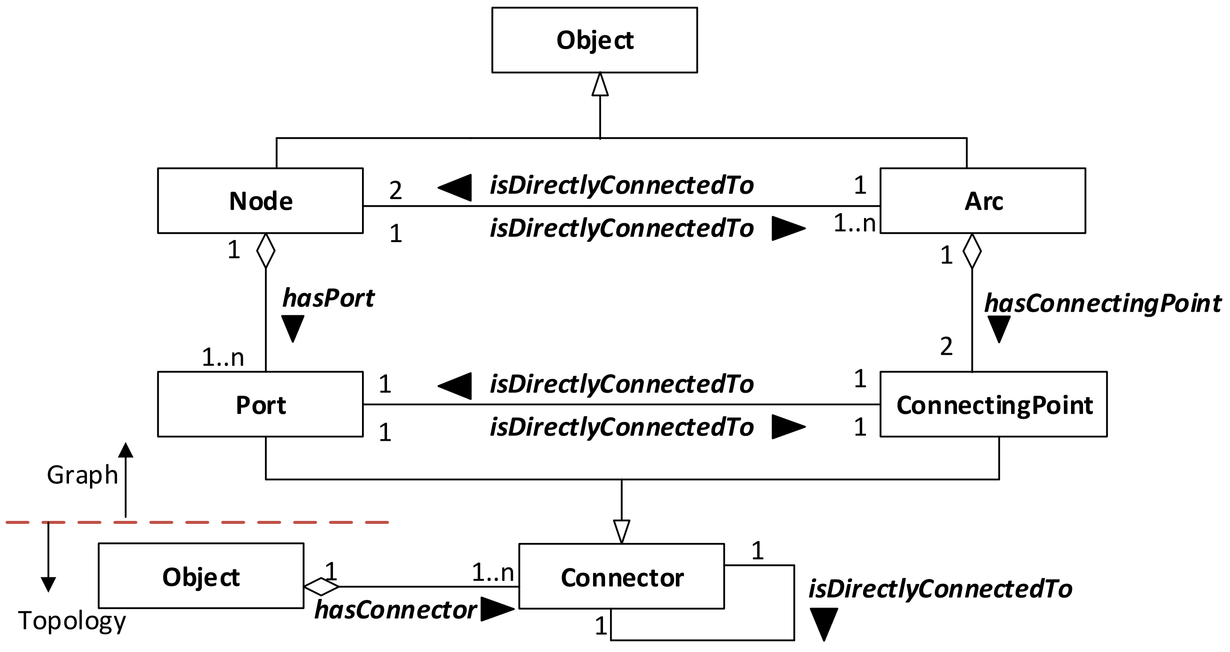


Figure 3.9: The Graph Information Model

In order to model the Graph Information Model, the connection or topology information should be added to the **Object** class first, which is being represented in the lower half of figure 3.9. The type and number of connections that an **Object** may have can be constrained by means of the **Connector** class. A **Connector** represents the interface through which an **Object** can be connected to another. Thus, an instance of the **Object** class should aggregate one or more instances of the **Connector** class. Such modeling requirements align with the design pattern used in the Part-Whole Information Model (the **Object** class mimics the **Whole** class, whereas the **Connector** class mimics the **Part** class) and thus the modelling mechanism between **Object** and **Connector** is same as that defined in the Part-Whole sub model. On top of that, it is mandatory that a **Connector** instance is connected to another **Connector** instance through ***isDirectlyConnectedTo*** object property.

The upper portion of the figure 3.9 further extends the connection or topology information to allow for the representation of graphs. The major concepts in the model are the **Node** class and the **Arc** class. Basically, an **Arc** cannot connect to more than two **Nodes**, which excludes arcs that fork. A **Node**, on the other hand, can be connected to one or more **Arcs**.

Also, a **Node** may have a list of **Ports**, whereas an **Arc** should have exactly two **ConnectingPoints**. Both of the **Port** class and the **ConnectingPoint** class denotes the interface information related to the **Node** class and **Arc** class. Thus they are modelled as the specializations of the **Connector** class. Also, the **Port** class and the **ConnectingPoint** class are related to each other through ***isDirectlyConnectedTo*** object property.

An application example of the Graph Information Model is shown in figure 3.10 below. A graph example with two nodes (*A* and *B*) is represented using the presented Graph Information Model. Both *node A* and *node B* have one port (*port_A_1* and *port_B_1* respectively) that denotes the connection interface and they are directly connected to the interface of *arc A-B (connectingPoint A-B-1 and connectingPoint A-B-2).*

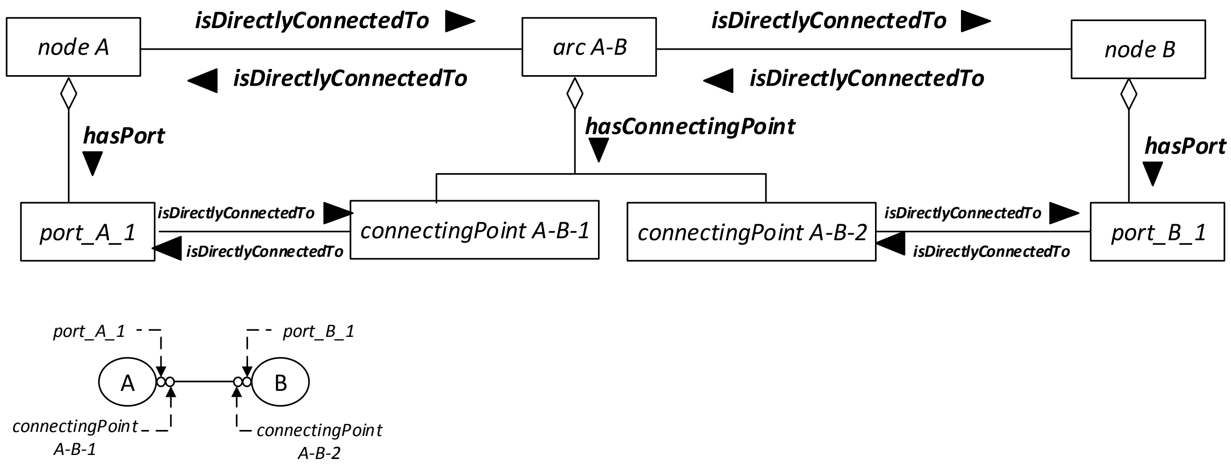


Figure 3.10: A Graph Application Example

(4) *System Model*

The last model in the abstract layer is the System Model and its objective is to provide a design pattern to represent different viewpoints of a complex system in a unified way. Systems are often too complex to be understood and handled as a whole. If we take an EOL product disassembly system as an example, related information could spread over several aspects like product, process, uncertainty, etc. We thus use a technique for complexity reduction that is widely used in the field of system engineering called the adaptation of viewpoint (Galster & Avgeriou, 2012). A viewpoint is an abstraction of the whole system restricted to a particular set of concerns. Adopting a viewpoint makes certain aspects of the system 'visible' while making other aspects 'invisible'. This way, we can focus on the specific viewpoints of a system, which is of special interest and address separately to the issues in other system viewpoints.

Figure 3.11 presents the System Information Model, which is extended based on the Part-Whole Information Model for representing the system hierarchy and the system decomposition. A **System** class is thus introduced as a subclass of the **Object** class and can be specified into either a **CompositeSubSystem** or an **AtomicSubSystem**. A **CompositeSubSystem** is a subsystem which can be further broken down into other subsystems, whereas an **AtomicSubSystem** is an elementary system that has no sub-systems of its own.

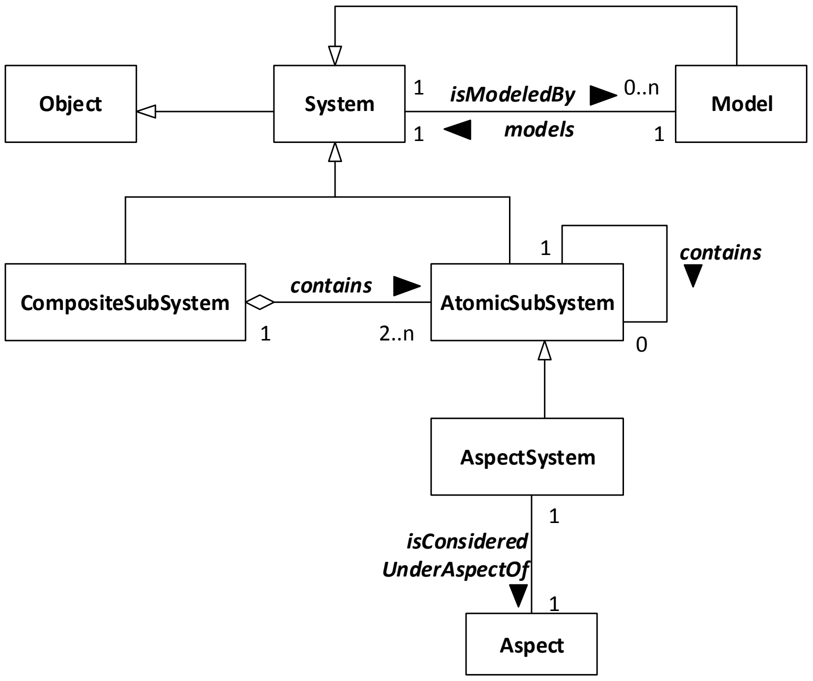


Figure 3.11: System Information Model

A special **System** called **Model** is also introduced here: a model is a system that is used or selected to enable the understanding the original system. In more detail, a **Model** could be used to resemble the physical object in a simplified way. An example could be an automotive mockup used for vibration testing. A **Model** could also represent the modeled system by means of some symbolic representation. Mathematical models or Information Models are the typical examples in this category. Following this definition, the class **Model** is introduced as a subclass of the **System** class (Figure 3.11). A **System** qualifies as a **Model** if it ***models*** some other **System.**

The last important concept in the System Information Model is called **AspectSystem**, which is used to denote different aspects about the overall system that are relevant to a particular viewpoint.

AspectSystem is modeled as a subclass of the **AtomicSubSystem** class and is related to the **Aspect** class through ***isConsideredUnderAspectOf*** object property.

The concept of **AspectSystem** plays a fundamental role in the modelling of the complex disassembly planning system. A general idea is presented in figure 3.12 below: The **DisassemblyPlanningSystem** class is being modeled as a specialization of the **CompositeSubSystem** class, which contains several **AspectSystems** (**Product, Process, Uncertainty** and **Degradation**). Each of the **AspectSystem** models the **DisassemblyPlanningSystem** under specific viewpoint (**Aspect**) and is a standalone sub model. As an example, the **Product** class is a subclass of **AspectSystem**, which targets on modeling the structural viewpoint of the disassembly system. Similarly, the **Process** subclass focuses on how to carry out each disassembly steps in order to achieve a certain component detachment task, thus it describes the behavior aspect of the overall disassembly system. The whole **DisassemblyPlanningSystem** is an aggregation of the four **AspectSystems.** The advantage of this design pattern is that the aspect systems can be used and maintained independently of the overall system.

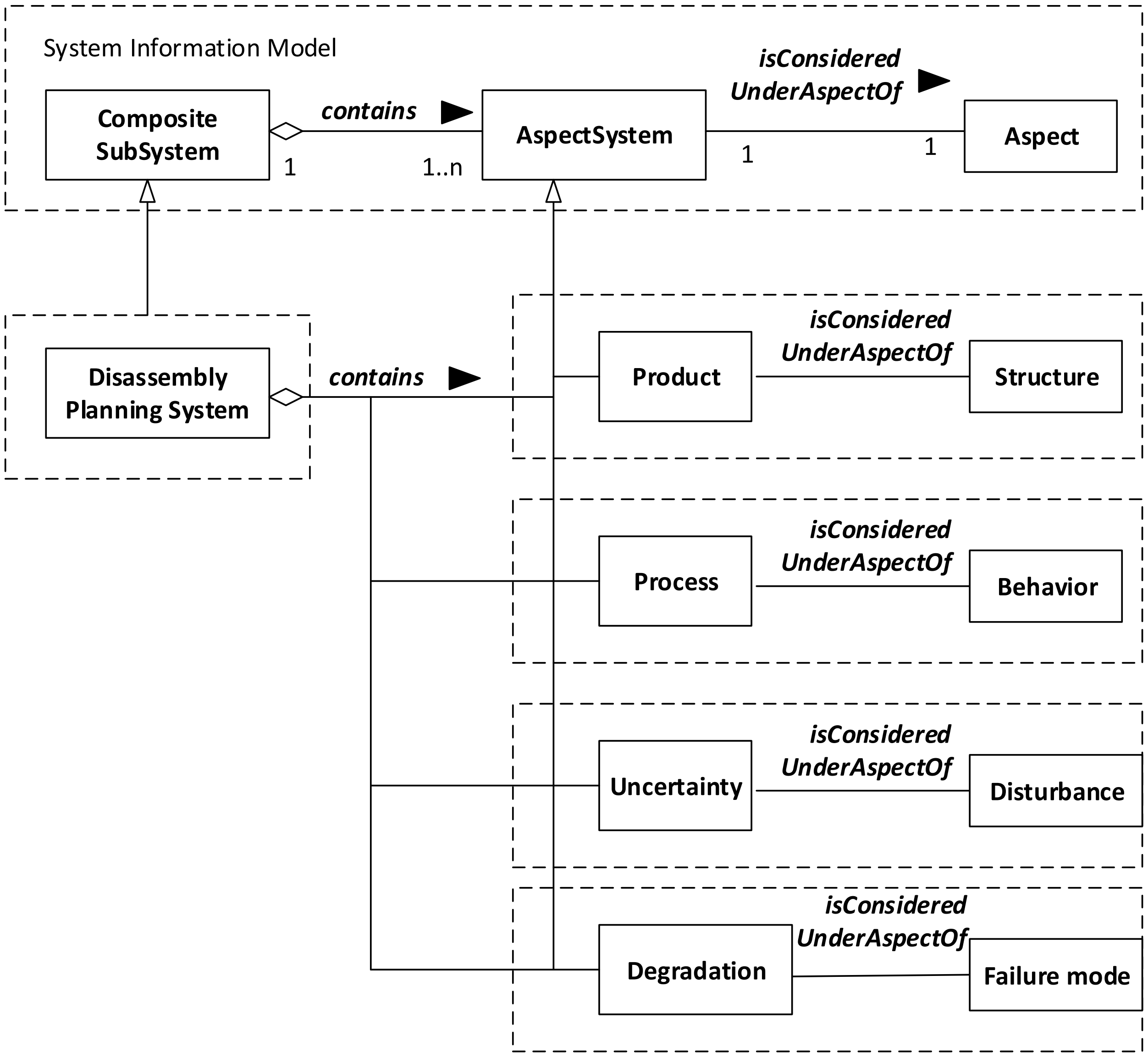


Figure 3.12: Aspect System Utilization in the Modelling of the Disassembly Planning System

3.3.2 Domain Layer Models

The Information Models in the domain layer capture the knowledge related to the domain of disassembly planning. Results from the disassembly planning requirement analysis (section 3.1) indicate that information from four aspects (product, process, uncertainty and degradation) should be included in the domain layer Information Models and each of them is modelled as a subclass of the **AspectSystem** class**,** which represents a standalone sub-model representing a specific viewpoint of the overall disassembly planning system. The following sections describe these domain layer sub-models in detail.

(1) Product Model

The Product Model is shown in figure 3.13 and it is being imported into the disassembly planning model to address the product aspect information requirements (refer to table 3.1 for detail). We will present the Product Model according to the different modelling requirements identified.

Modelling of the System Aspect

The **Product** class represents the EOL product which is under study and it is being modeled as the subclass of the **AspectSubSystem** class. It means that a product is considered as a subsystem which represents a specific aspect (structure) of the overall disassembly planning system.

Modelling of Product Hierarchy (Part-whole relationship)

A certain EOL product contains one or more different components or subassemblies which are organized in a hierarchical order and such requirement is achieved by introducing the **SubAssembly** class and the **Component** class. The design pattern in the Part-Whole sub model in the abstract layer is used here for modelling the parthood relationships among the product, the subassembly and the component. Specifically, the **Product** class aggregates the **SubAssembly** class and the **Component** class through the object properties ***hasSubAssembly*** and ***hasComponent***. Similar parthood relationship exists between the **SubAssembly** class and the **Component** class: an instance of the **SubAssembly** class contains at least two instances of the **Component** class and the object property ***hasComponent*** is used to address such relationship. Last but not least, it is possible that an instance of the **Component** class can contain more than one other **Component**. This will be described in detail when describing the classification of the **Component** class.

Component Classification

In the domain of disassembly planning, further specifications of the **Component** class according to their characteristics are important. As it is described in the previous sections, the **Component** class can be further classified according to the material composition (homogeneous or composite), functional type (connecting function or non-connecting function) and component complexity (atomic component or complex component). Thus, several **Component** subclasses are introduced as follows:

ConnectingComponent is a subclass of the **Component** class whose primary function is to connect other components. Examples of the **ConnectingComponent** can be screws, insert pins, etc. The **VirtualConnectingComponent** class is further introduced as a special type of **ConnectingComponent**, which is a virtual component used for disassembly planning. As an example, a Velcro connection is a common type of connection in which no **ConnectingComponent** is involved, but rather use self-engaging loops to achieve the binding between components. In order to carry out disassembly planning analysis consistently for such cases, the concept of **VirtualConnectingComponent** is introduced to emulate a virtual connecting component between the connected components.

ConnectingComponent is the candidate component to be analyzed when carrying out the disassembly process and it can be detached at least from one direction. In other words, a **ConnectingComponent** contains at least one **DegreeOfFreedom**.

The **OrdinaryComponent** class represents all the components other than the **ConnectingComponent**, whose primary function is not to connect other components. Examples of the **OrdinaryComponent** can be the blender housing, the coffee maker jar, etc. for a blender machine. Different from the **ConnectingComponent, OrdinaryComponent** is fully constrained on all directions in the beginning of the disassembly process (we don't consider movable **OrdinaryComponent** in this thesis). Thus, an instance of the **OrdinaryComponent** class contains zero **DegreeOfFreedom**.

The **OrdinaryComponent** class can be further classified into the **AtomicOrdinaryComponent** class and the **ComplexOrdinaryComponent** class. The **ComplexOrdinaryComponent** is an **OrdinaryComponent** which contains a set of irreversibly connected components. Examples of such component can be electrical cables, printed circuit board, etc. The **AtomicOrdinaryComponent**, on the other hand, represents the most elementary component and contains no other component.

The last level of the component classification is related to the material composition (homogeneous or composite), and depending on the number of the different types of homogeneous materials an **OrdinaryComponent** contains**,** the **OrdinaryComponent** class can be further classified into either the **HomogeneousOrdinaryComponent** class or the **ComplexOrdinaryComponent** class.

Product Topology (Component Liaison)

The final requirement in the Product Model is to model the EOL product topology and the component liaison. Since the connection among components in an EOL product can be viewed as a graph: node represents components, whereas connection between components is represented as edges between nodes. The Graph Model in the abstract layer is thus used to model the product structure. Figure 3.14 shows the comparison between the graph model and the relevant classes in the Product Model for the modelling of the product topology.

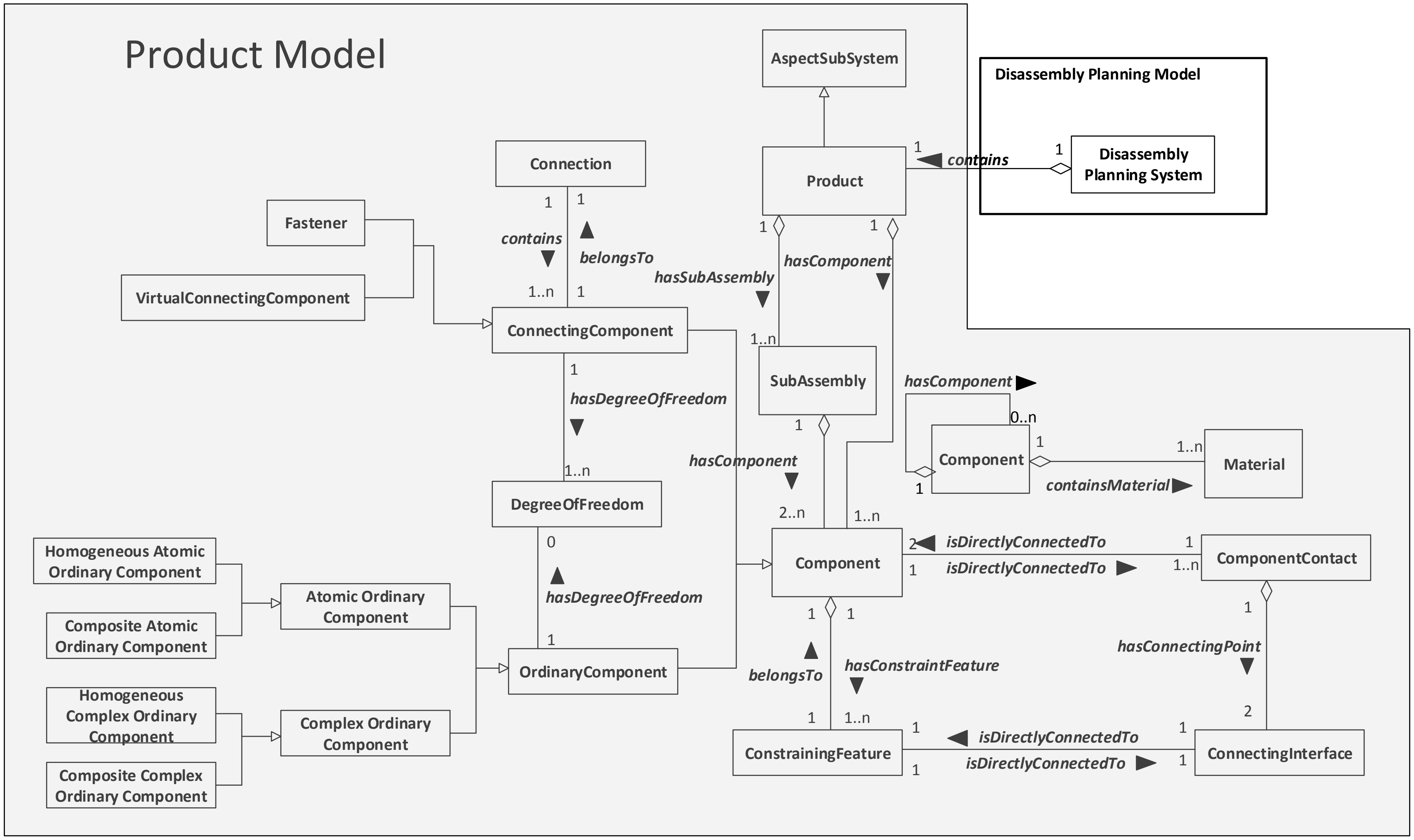


Figure 3.13: Product Information Model for Disassembly Planning

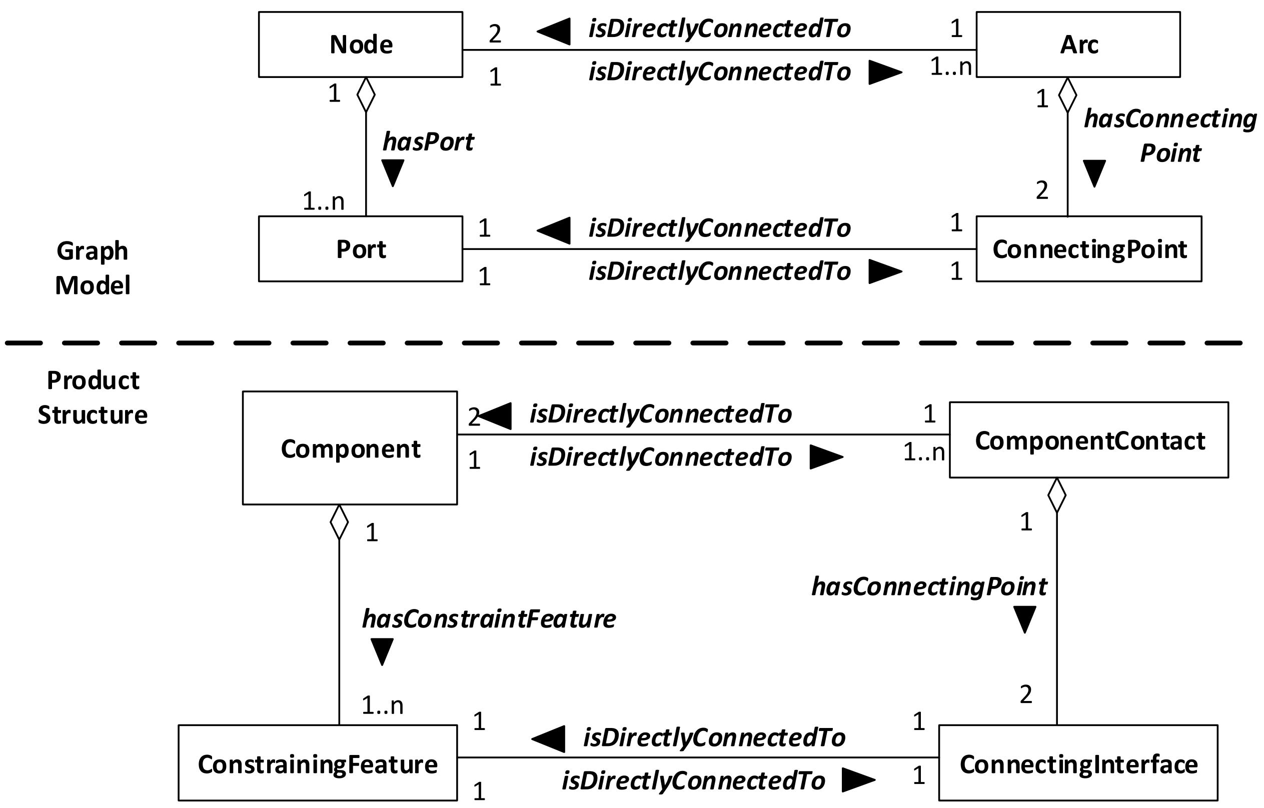


Figure 3.14: Comparisons between the Graph Model and the Product Model

An instance of the **Component** class, similar to the **Node** class in the Graph Model, is directly connected to one of more instances of the **ComponentContact** class, which is comparable to the **Arc** class in the Graph Model. Also, an instance of the **ComponentContact** class is directly connected to exactly two **Component** instances.

Both of the **Component** class and the **ComponentContact** class contain some interfaces, through which they connect to the each other. Such interface information is implemented by introducing the **ConstrainingFeature** class and the **ConnectingInterface** class respectively. The **ConstrainingFeature** class represents the component feature (face, edge or face) which has direct contact with the features in the connected **Component**, whereas the **ConnectingInterface** class is comparable to the **ConnectingPoint** class in the Graph Model and represents the interface of the **ComponentContact** class.

An example of three connected components (*Component_A, Component_B and Component_C*) is shown in figure 3.15 below. *Component_A, Component_B and Component_C* are instances of the **Component** class and they contain certain features which is directly involved in a connection. As an examples, *A-f1* is the bottom planer surface of *Component_A* and it is modeled as an instance of the **ConstrainingFeature** class, which means that *A-f1* is the port or interface through which *Component_A* is connected to the other component (*Component_B* in this case).

Similarly, we have *ComponentContact_1*, *ComponentContact_2* and *ComponentContact_3* being modeled as the instances of the **ComponentContact** class and their role is similar to the role of the arc in the Graph Model. Lastly, each **ComponentContact** instance holds exactly two interface objects (**ComponentInterface**), through which it connects to the components. In the example in figure 3.15, *ComponentContact_1* is directly connected to *CI_1* (*ComponentInterface_1*) and *CI_2* (*ComponentInterface_2*).

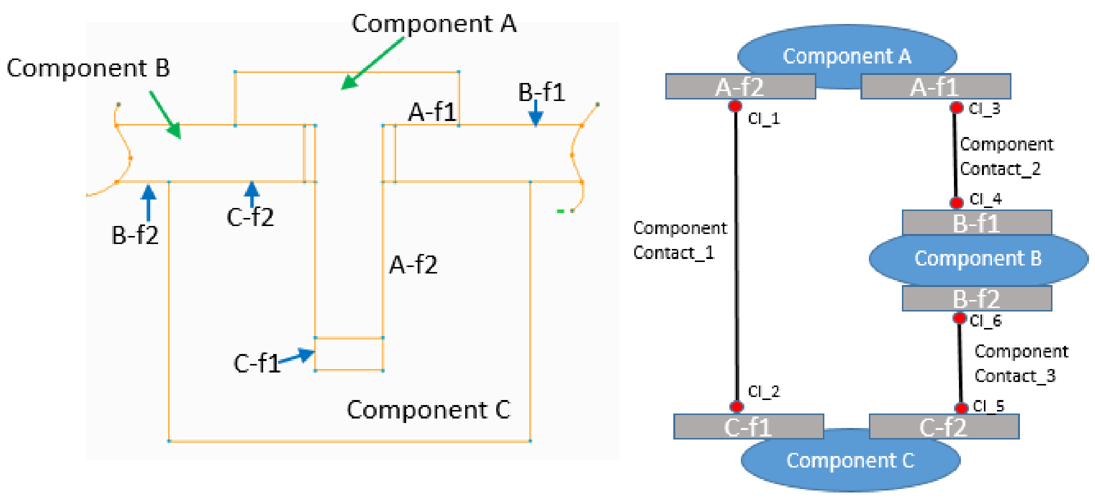


Figure 3.15: An Example of the Product Topology

(2) Process Model

The disassembly process accomplishes the basic transformations of the product states in the domain of disassembly and describes how disassembly of an EOL product can be achieved (i.e. the behavior of the disassembly planning system). Similar to the Product Model, the Process Model is also considered as a special sub system (subclass of the **AspectSystem** class), whose primary function is to address the process related information requirements in the overall disassembly planning system. Such requirements have been analyzed in section 3.1 and two major requirements have been identified: (1) the modeling of the process hierarchy and (2) the modeling of all feasible disassembly process sequences (refer to table 3.2 for detail). Figure 3.16 shows the overall structure of the Process Model, which addresses these process related information requirements in detail.

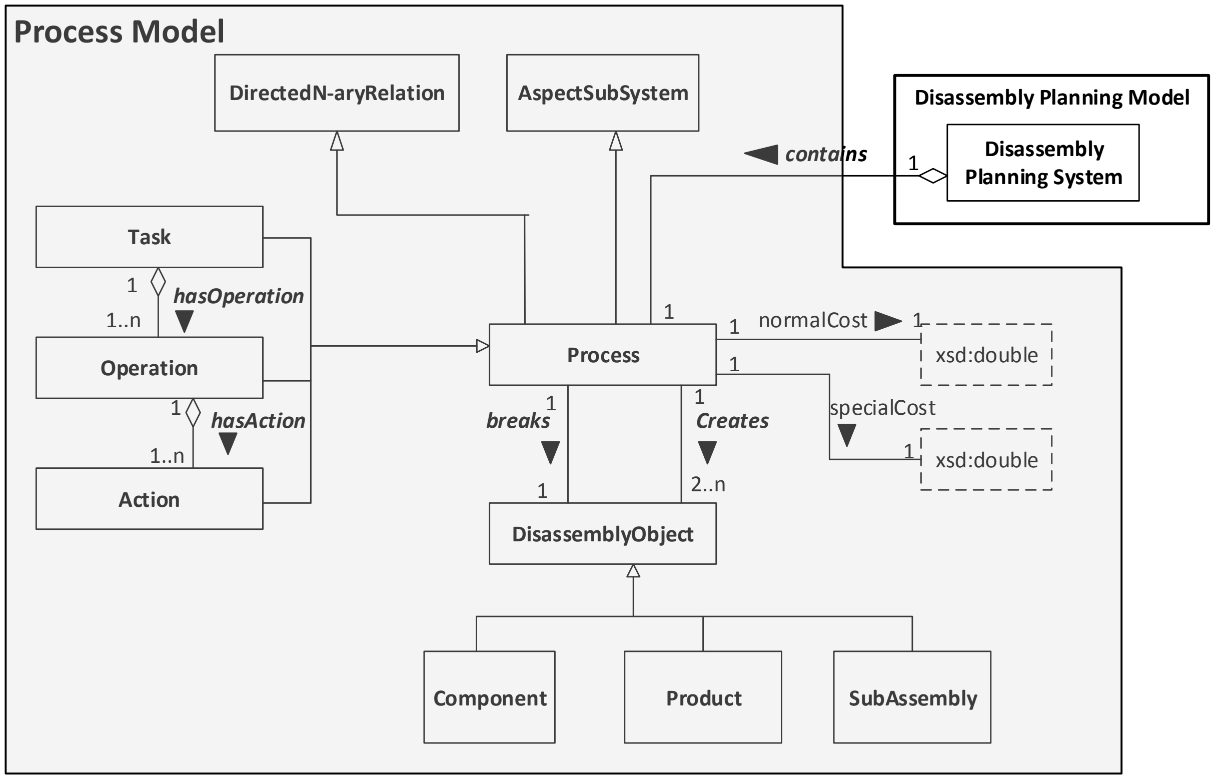


Figure 3.16: The Overall Structure of the Process Model

Modeling of the process hierarchy

A disassembly **Process** can be analyzed from different hierarchical abstractions. Three types of disassembly process are thus introduced: the disassembly **Task**, the disassembly **Operation**, and the disassembly **Action** (refer to section 3.1.2 for detailed descriptions of these concepts). Furthermore, parthood relationships exist among the **Task** class, the **Operation** class and the **Action** class: an instance of the **Task** class contains one or more instances of the **Operation** class and an instance of the **Operation** class contains one or more instances of the **Action** class.

An example of such a process hierarchy is illustrated in figure 3.17. The most abstract disassembly process description resides at the task level, which only specifies the target component to be detached at a certain disassembly stage. Such high level task ("*Detaching Component A"* in this example) is further decomposed into three operation level processes (*"Orientation of Product"*, *"Tool Change" and "Disassembly of Component A"*) which are necessary in order to achieve the "*Detaching Component A*" task. Similarly, the operation level process can be further specified, which presents the most concrete elemental action level disassembly process.

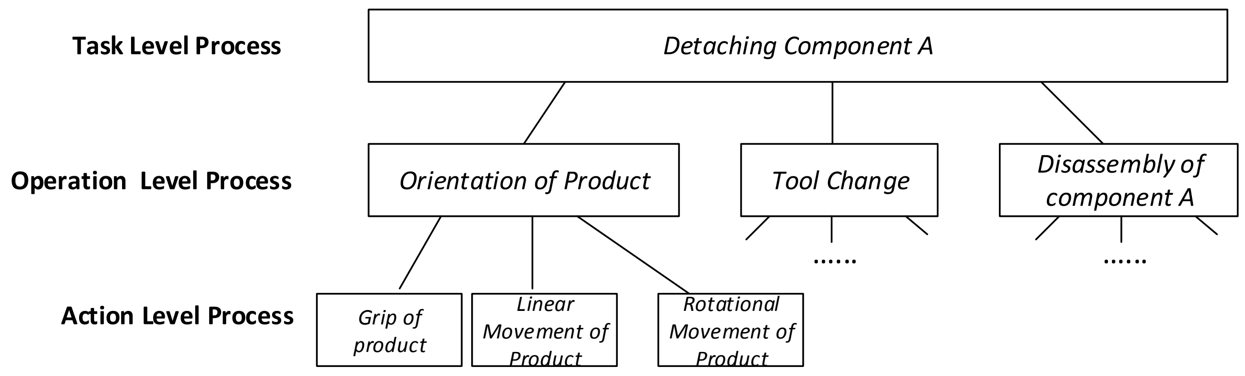


Figure 3.17: Disassembly Process Decomposition Example

Modelling of all the feasible disassembly process sequences

The second requirement for the process model is to represent all the feasible process sequences, which is a critical input information for the disassembly sequence optimization. Even though the disassembly process can be analyzed under different abstraction levels, they can all be treated as a type of N-ary relationship among different disassembly objects (Component, Subassembly or Product). Let us take the AND/OR graph in figure 3.18 as an example; each edge represents a disassembly process (the process is at the task level in this example) which involves at least three disassembly objects: One **Process** breaks one **DisassemblyObject** which represents the pre-state of the disassembly process and in the same time creates two or more disassembly objects which represents the post-state of the disassembly process. In this example, *task_1* is an instance of the **Process** class which breaks the **DisassemblyObject** *ABCDE* (**Product**) and creates the **DisassemblyObject** *A* (**Component**) and the **DisassemblyObject** *ABCD* (**Subassembly**). The whole disassembly process sequence thus is an aggregation of the instances of the **Process** class.

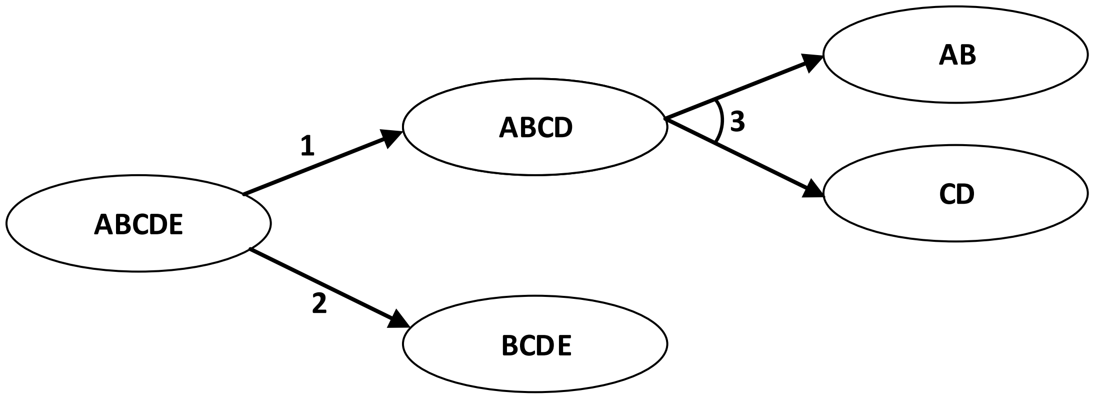


Figure 3.18: AND/OR Graph of a Product

The N-ary relation model in the abstract layer is used to model such modeling requirement. The mechanisms to model the feasible disassembly process is analogous to the design pattern as defined in the N-ary relation model (figure 3.19) with minor extensions. In detail, the **Process** class is being modeled as an extension of the **DirectedRelationship** class and it relates to the **DisassemblyObject** class through two object properties: ***breaks*** and ***creates***, which are comparable to the object properties defined in the N-ary relation Model (***hasOrigin*** and ***hasTarget***). Similarly, two process attributes normalCost and specialCost are introduced to represent the cost related information. This is comparable to the realtionAttribute data property introduced in the N-ary relation Model.

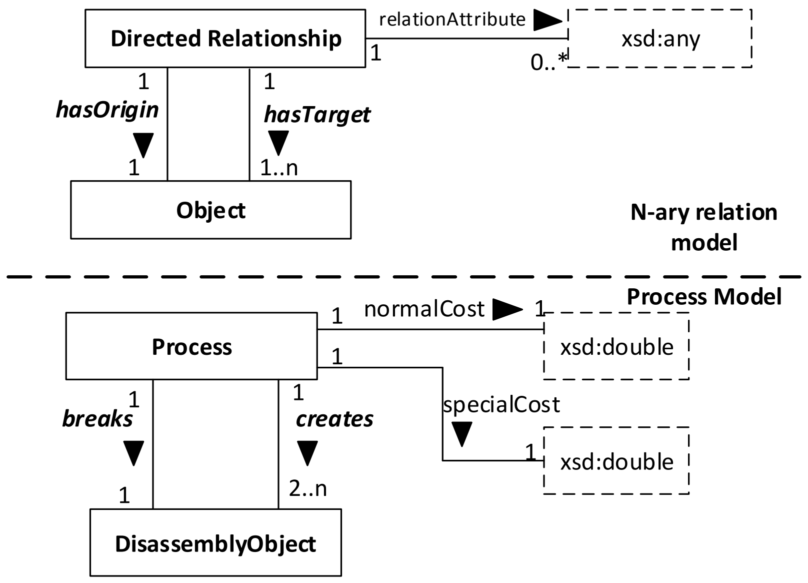


Figure 3.19: Comparison between Process model and N-ary Relation Model

(3) Uncertainty Model

Unlike the assembly process, the disassembly process has various uncertainty issues. Two types of uncertainty are considered in this dissertation: (1) Component/subassembly/product functional uncertainty and (2) the process uncertainty. Both cases are handled using the Bayesian theorem. The disassembly uncertainty handling procedure is presented in detail in chapter 5. Here, we only focus on modeling the required related information. Figure 3.20 presents the overall structure of the Uncertainty Model.

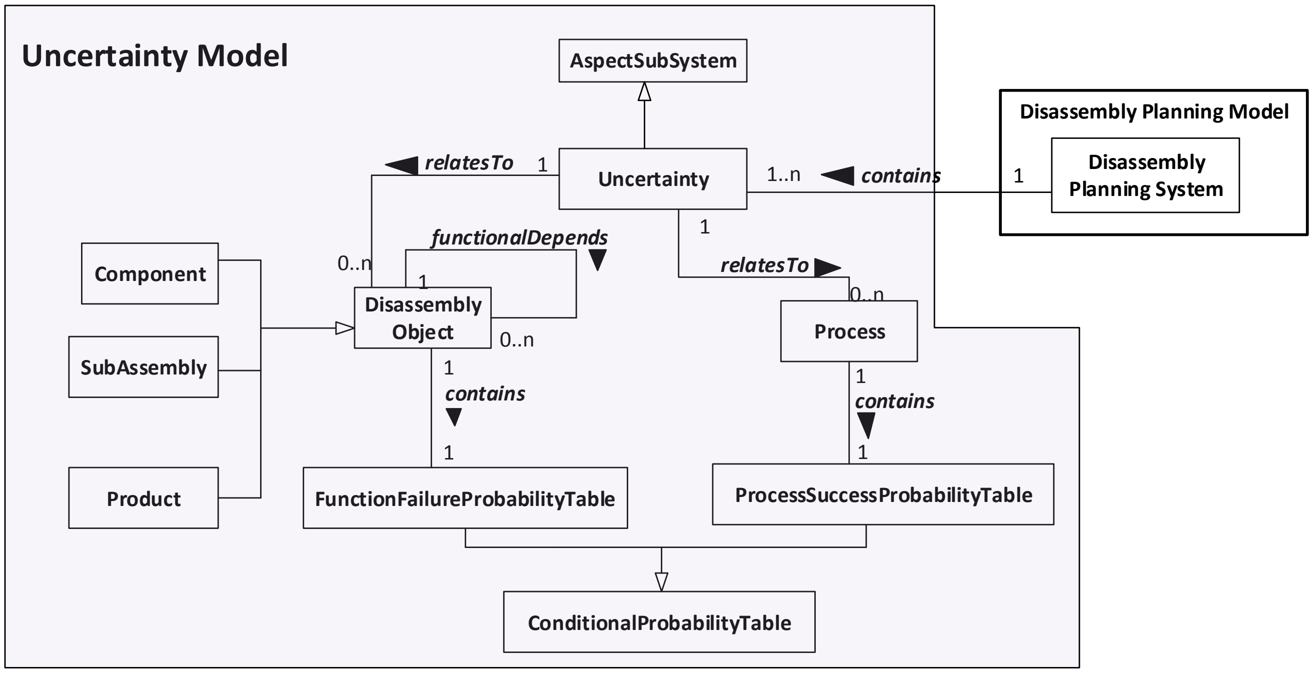


Figure 3.20: The Structure of the Uncertainty Model

First of all, like the previous domain layer models, the **Uncertainty** class is being modeled as the subclass of the **AspectSubSystem** class, whose primary focus is on the modelling of the information related to the disturbance aspect in a disassembly planning system.

The **Uncertainty** class relates to two classes (**DisassemblyObject** and **Process**), which represent the two specific uncertainty issues (component/subassembly/product functional uncertainty and process uncertainty) studied in this thesis. The **DisassemblyObject** class is introduced to represent the aggregation of the **Product**, the **SubAssembly**, and the **Component** instances. The functional status of a **DisassemblyObject** instance might be dependent on the functional status of the other **DisassemblyObject** instances and such a situation is realized by the object property ***functionalDepends.*** As an example, "prepare food" is the function of a blender (**Product**) and whether an old blender can function properly is conditionally dependent on the other

DisassemblyObjects contained in the blender (in this case, the functional status of the Motor subassembly, of the central control unit, etc.).

Both of the **DisassemblyObject** class and the **Process** class contain a conditional probability table which is used for carrying out uncertainty reasoning and such information is being modeled in the **FunctionFailureProbabilityTable** class and the **ProcessSuccessProbabilityTable** class accordingly.

(4) Degradation Model

Degradation is also a critical issue in the planning of disassembly. It usually refers to a gradual change in the properties (like tensile strength, color, shape, etc.) of the component or subassembly, which usually does not affect the overall function of the component/subassembly until reaching a critical point. From the analysis in section 3-1, we know that the existence of degradation in the EOL product has a lot of influences on the economic quantification of EOL products or components, which is a critical issue in the disassembly planning process. The Degradation Model is thus introduced to represent the relevant information necessary for the degradation analysis. In this thesis we use the fuzzy logic approach to model the component/subassembly degradation and quantify their reuse value through a set of fuzzy linguistic variables and a set of heuristic rules. The procedure for reuse value estimation using fuzzy logic is presented in detail in chapter 5. Here we only summarize the important informational elements necessary for carrying out the fuzzy logic based reuse value estimation. Figure 3.21 presents the overall structure of the Degradation Model.

Similar to the other domain level Information Models, the class **Degradation** is being modeled as a sub class of the **AspectSystem** class, which represents the information related to the failure mode of the disassembly objects (component, subassembly or product) in an EOL product. Like the definition used in the Uncertainty Model, we introduce the class **DisassemblyObject** to represent the aggregation of the **Component** class, the **Subassembly** class and the **Product** class. One **DisassemblyObject** instance relates to several **FuzzyVariable** instances, which are used to infer the reuse value of the **DisassemblyObject**. Each **FuzzyVariable** comprises of several other information including the type of the variable (input or output), lower limit, upper limit and a set of **FuzzyTerms**. Each **FuzzyTerm** further comprises of information like the name of the fuzzy term, type of the membership function and the parameters for defining the membership function.

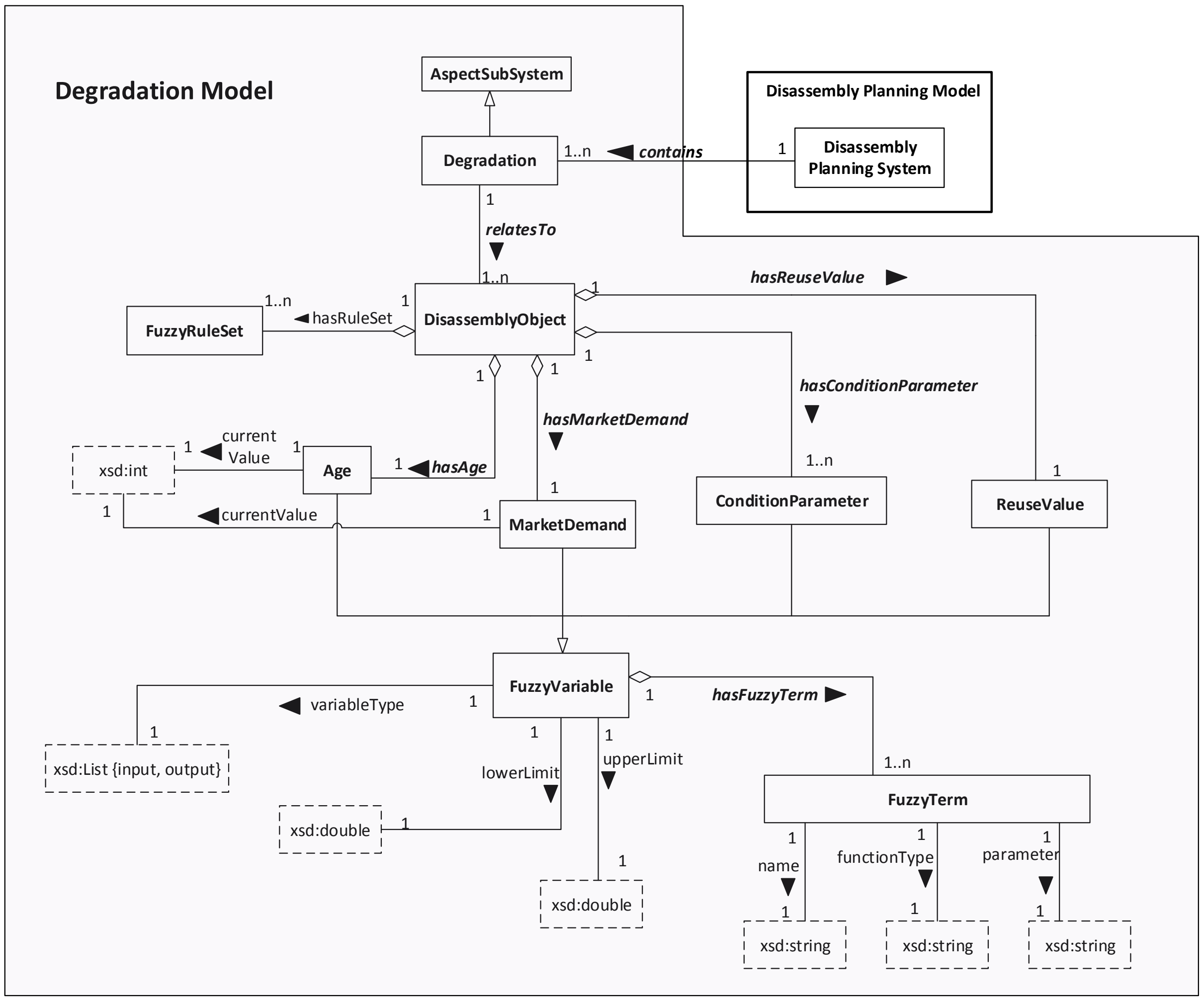


Figure 3.21: Overall Structure of the Degradation Model

Four types of fuzzy variable (**Age**, **MarketDemand**, **ConditionParameter** and **ReuseValue**) are identified in section 3-1 and they are being modeled as the sub class of the **FuzzyVariable** class. Among them, the classes **Age**, **MarketDemand**, **ConditionParameter** are the fuzzy input variable whereas the class **ReuseValue** is the fuzzy output variable. Lastly, each **DisassemblyObject** contains a set of fuzzy rules to demonstrate the heuristic relations among the fuzzy variables. These are defined in the **FuzzyRuleSet** class.

3.4 Formal DIM Implementation based on Web Ontology Language

DIM represents a conceptual information model in the domain of disassembly planning and it has been implemented into the Web Ontology Language (OWL) for formal machine reasoning and interpretation. Before presenting the DIM OWL implementation, the background and reason to use OWL as the implementation language is briefly discussed below.

3.4.1 Why using OWL for DIM implementation

The implementation of DIM is under the paradigm of Internet of Things (IoT) and the concept of "Life Cycle Unit" (LCU). Briefly, IoT provides a network to connect different physical objects, which allows them to be sensed and controlled remotely across existing network infrastructure, creating opportunities for more direct integration of the physical world into computer-based systems, and resulting in improved efficiency, accuracy and economic benefit. LCU, on the other hand, is developed specifically for the product disassembly process. As mentioned before, in a disassembly factory, different products arrive continuously for disassembly, and individual decisions regarding optimal disassembly sequences have to be made for every product. It is difficult to predict any pre-defined disassembly process sequences a priori, so the detailed information on how to disassemble each arriving product is needed. LCU is proposed under the idea of decentralizing that information by integrating a physical device named Life Cycle Units (LCU) into every product. The LCU stores information needed for disassembly. Once enough disassembly information about a product is present, the optimal disassembly sequence can be generated based on the actual physical status of the EOL product. Combining the LCU and IoT technologies together, individualized EOL product information could be sensed and collected by LCU and transferred to the central Product Lifecycle Management (PLM) system through the IoT network. Now, disassembly researchers could have the potentials to tackle the problem of disassembly information bottleneck.

The implementation of DIM thus should support the above new paradigm and it should have the ability to (i) sensing the environment, (ii) store the data in the digital memory, (iii) communicate with other facilities or smart products and (iv) carry out knowledge reasoning. This gives new opportunities for solving the problem of EOL product disassembly in which product life cycle data plays a fundamental role. In this dissertation, we focus on the information aspect and three important related requirements (with OWL solutions) are listed below:

R1: LCU usually requires a fast processing with restricted resources and thus the DIM implementation syntax needs to be compact.

Solution: OWL file is an XML based textual file, which can be processed or reasoned by lightweight existing query and reasoning plugins.

R2: The implemented DIM is going to be published as a formal Disassembly Information Model on the Web with the possibility to be connected to the other domain applications. Thus, the implementing language should provide easy mechanisms for connecting DIM with other “things” on the Internet without pre-assumptions.

Solution: In OWL, each object defined in the DIM is annotated with a Unified Resources Identifier (URI), which helps other applications to access the information and connect to DIM through the web. On top of that, OWL does not follow the traditional database approach, it adopts the open world assumption (the truth value of a statement may be true irrespective of whether or not it is known to be true), which facilitates the further extensions of the proposed DIM: users can add knowledge to DIM as long as it did not semantically contradict with the current definitions and such a validation process can usually be carried out automatically by certain reasoners like Pellet or Hermit.

R3: The implemented DIM should facilitate the automated knowledge reasoning process.

Solution: Semantic Web Rule Language (SWRL) is supported by OWL, which extends the set of OWL axioms to include Horn-like rules. It thus enables Horn-like rules to be combined with an OWL knowledge base to provide flexible and powerful knowledge reasoning capabilities.

3.4.2 OWL Implementation

Protégé 4.0 (Gennari et al., 2003), as an OWL editor, is used in this thesis to develop formal DIM OWL implementation. The complete OWL implementation code can be accessed from: http://disassembly-planning-ontology.sourceforge.net. Here, several OWL implementation issues are discussed in detail and full summarization of DIM implementation is presented in the appendix.

(1) Model hierarchy

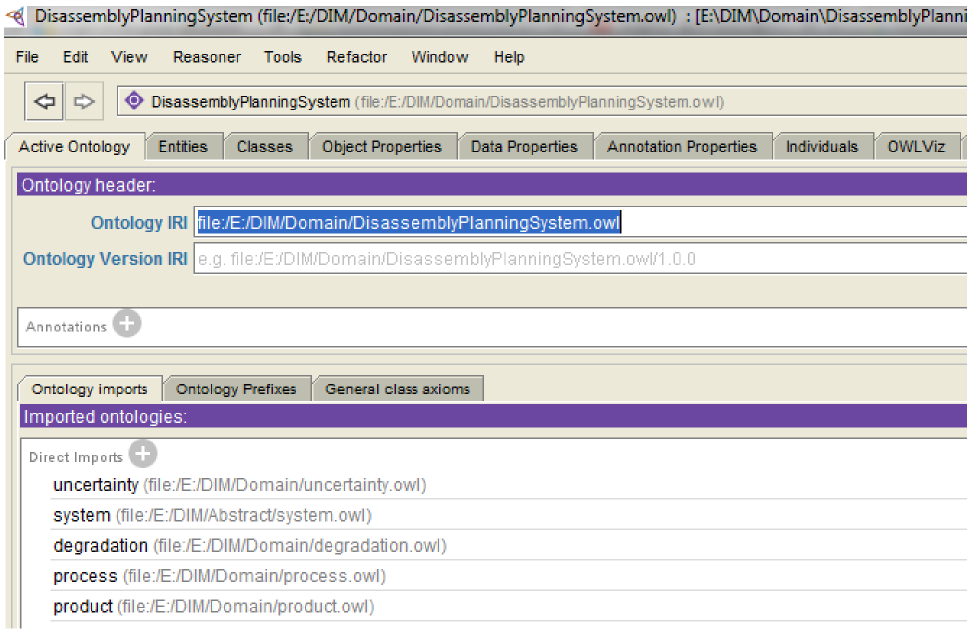
The DIM is a set of models distributed in three hierarchical layers, and certain dependence or aggregation relationship exists among different sub models. As an example, the "DisassemblyPlanningSystem" sub model, being the most complex sub model in the domain layer, is actually an aggregation of four domain sub models (Product Model, Process Model, Uncertainty

Model and Degradation Model). On top of that, the “DisassemblyPlanningSystem” model itself is a system in nature, thus it is dependent on the System Model from the abstract layer.

DIM is thus implemented into 11 OWL sub models shown in table 3.5. Each OWL file represents a sub model in the DIM and is classified into different layers (abstract layer, domain layer or application layer). The dependence or aggregation relationship exists among models is realized by importing the related sub models into the existing model. Figure 3.22 shows a screenshot of OWL implementation of the “DisassemblyPlanningSystem” sub model, which imports 5 sub models it depends on.

Table 3.5: DIM OWL Implementation

Abstract Layer	***Domain Layer***	**Application Layer (Chapter 4 & 5)**
N-ary-relationship.owl	DisassemblyPlanning System.owl	DisassemblySequenceGenerator.owl
Part-whole.owl	Product.owl	AdaptiveDisassemblyPlanner.owl
Graph.owl	Process.owl	
System.owl	Uncertainty.owl	
	Degradation.owl	


DisassemblyPlanningSystem (file:/E:/DIM/Domain/DisassemblyPlanningSystem.owl) : [E:\DIM\Domain\DisassemblyPlanni

DisassemblyPlanningSystem (file:/E:/DIM/Domain/DisassemblyPlanningSystem.owl)

Ontology header:

Ontology IRI file:/E:/DIM/Domain/DisassemblyPlanningSystem.owl

Ontology Version IRI e.g. file:/E:/DIM/Domain/DisassemblyPlanningSystem.owl/1.0.0

Imported ontologies:

Direct Imports

uncertainty (file:/E:/DIM/Domain/uncertainty.owl)
system (file:/E:/DIM/Abstract/system.owl)
degradation (file:/E:/DIM/Domain/degradation.owl)
process (file:/E:/DIM/Domain/process.owl)
product (file:/E:/DIM/Domain/product.owl)

Figure 3.22: Importing Sub models into the Dependent Model

(2) Modeling the Class Relationship

A large amount of the elements in the DIM are meant to model the relationships among classes (specify how the individuals relate to other individuals). As an example, in the Process Model (figure 3.23), the relationships ***"breaks"*** and ***"creates"*** relate the **Process** class with the **DisassemblyObject** class and the relationship *normalCost* and *specialCost* relates the **Process** class with a certain datatype class (xsd: double). In OWL, the entities describing the ways individuals are related are called properties. Two types of properties can be further specialized: if the property describes one individual's relatedness to other individuals, like ***breaks*** and ***creates*** in the Process Model, it is called "object property"; if the property relates individuals to data values (instead of to other individuals), like the *normalCost* and the *specialCost* in the Process Model, it is called "datatype property". Figure 3.24 shows a screenshot of implementing the relationships existing in the Process Model in Protégé.

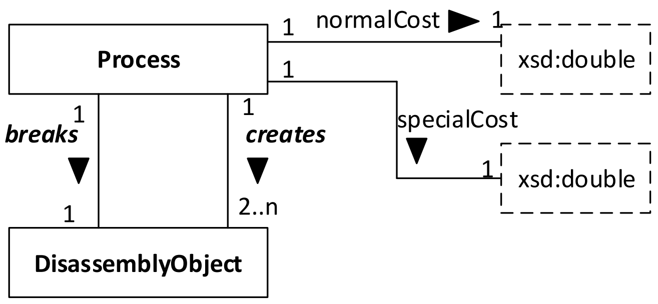
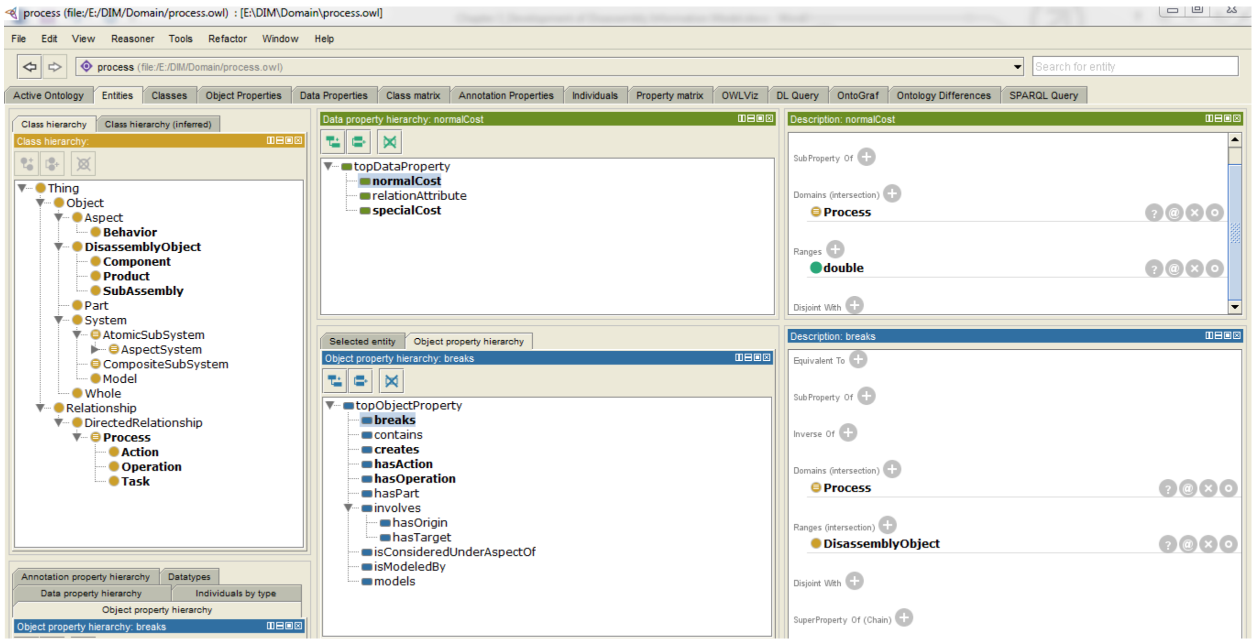


Figure 3.23: Relationship in the Process Sub Model



process (file:/E:/DIM/Domain/process.owl) : [E:\DIM\Domain\process.owl]
File Edit View Reasoner Tools Refactor Window Help
process (file:/E:/DIM/Domain/process.owl)
Search for entity
Active Ontology | Entities | Classes | Object Properties | Data Properties | Class matrix | Annotation Properties | Individuals | Property matrix | OWLViz | DL Query | OntoGraf | Ontology Differences | SPARQL Query
Class hierarchy | Class hierarchy (inferred)
Data property hierarchy: normalCost
Description: normalCost
Selected entity | Object property hierarchy
Object property hierarchy: breaks
Description: breaks
Annotation property hierarchy | Datatypes
Data property hierarchy | Individuals by type
Object property hierarchy

```
Class hierarchy:
▼ Thing
  ▼ Object
    ▼ Aspect
        Behavior
    ▼ DisassemblyObject
        Component
        Product
        SubAssembly
      Part
    ▼ System
      ▼ AtomicSubSystem
        ► AspectSystem
        CompositeSubSystem
        Model
      Whole
  ▼ Relationship
    ▼ DirectedRelationship
      ▼ Process
          Action
          Operation
          Task
```

```
Data property hierarchy: normalCost
▼ topDataProperty
    normalCost
    relationAttribute
    specialCost
```

```
Description: normalCost
SubProperty Of
Domains (intersection)
    Process
Ranges
    double
Disjoint With
```

```
Object property hierarchy: breaks
▼ topObjectProperty
    breaks
    contains
    creates
    hasAction
    hasOperation
    hasPart
  ▼ involves
      hasOrigin
      hasTarget
    isConsideredUnderAspectOf
    isModeledBy
    models
```

```
Description: breaks
Equivalent To
SubProperty Of
Inverse Of
Domains (intersection)
    Process
Ranges (intersection)
    DisassemblyObject
Disjoint With
SuperProperty Of (Chain)
```

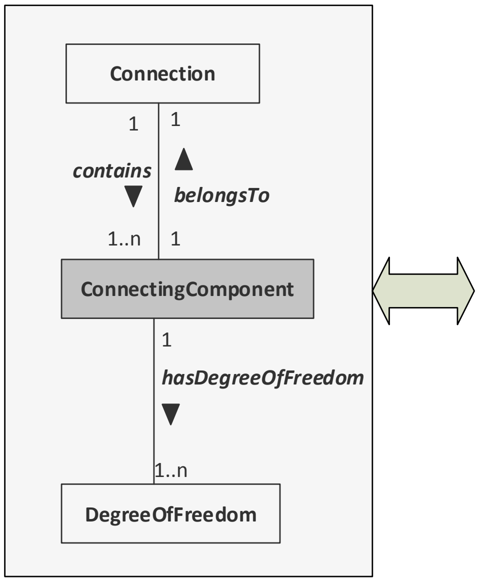
Figure 3.24: Implementing Object Property and Datatype Property

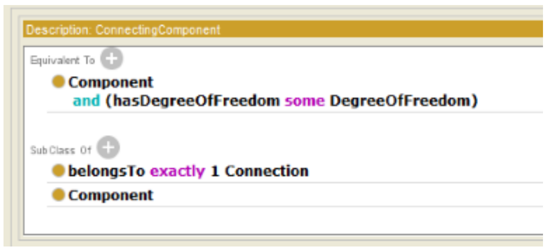
(3) Modeling of the Class Axioms

The last important issue in implementing the DIM is to include semantic axioms to represent the necessary and sufficient conditions of a certain class. As an example, in the Product Model, different types of the **Component** class are specified. Figure 3.25 shows one type of the **Component** in the component taxonomy named **ConnectingComponent**, which consists of two important axioms to define the necessary and sufficient conditions that the **ConnectingComponent** class holds:

A1 (Sufficient condition): **ConnectingComponent** is a **Component** which contains at least at one degree of freedom, through which the **ConnectingComponent** can be detached.

A2 (Necessary condition): A **ConnectingComponent** instance ***belongsTo*** certain type of the **Connection**. E.g. A screw is an instance of the **ConnectingComponent** class and it ***belongsTo*** the screw connection.




Description: ConnectingComponent

Equivalent To

Component
and (hasDegreeOfFreedom some DegreeOfFreedom)

SubClass Of

belongsTo exactly 1 Connection

Component

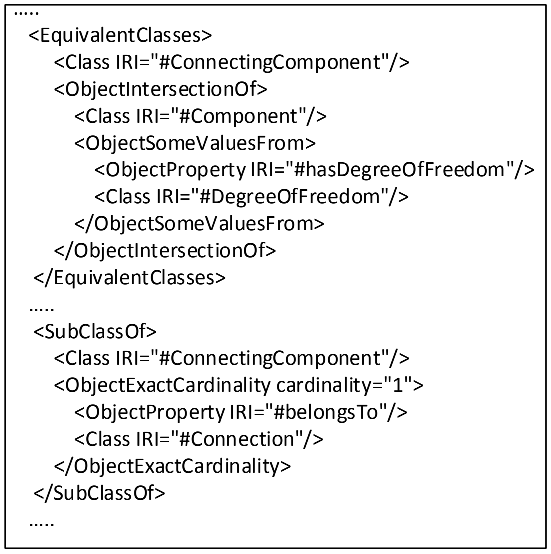

```
.....
 <EquivalentClasses>
   <Class IRI="#ConnectingComponent"/>
   <ObjectIntersectionOf>
     <Class IRI="#Component"/>
     <ObjectSomeValuesFrom>
       <ObjectProperty IRI="#hasDegreeOfFreedom"/>
       <Class IRI="#DegreeOfFreedom"/>
     </ObjectSomeValuesFrom>
   </ObjectIntersectionOf>
 </EquivalentClasses>
 .....
 <SubClassOf>
   <Class IRI="#ConnectingComponent"/>
   <ObjectExactCardinality cardinality="1">
     <ObjectProperty IRI="#belongsTo"/>
     <Class IRI="#Connection"/>
   </ObjectExactCardinality>
 </SubClassOf>
 .....
```

Figure 3.25: Implementing the Semantic Axioms related to the Class ConnectingComponent

Both of the axioms can be implemented formally in OWL and they are shown on the left side of figure 3.25. Generally, they follow the following syntax:

The sufficient condition is defined under the <EquivalentClasses> tag, formally as:

```
<axiom>::= EquivalentClasses( <description> )
```

Similarly, the necessary condition is defined under the <SubClassOf> tag, formally as:

```
<axiom> ::= SubClassOf( <description> )
```

Where, <description> can further be aggregation of several items including classes, restrictions, etc. and the formal metadata for <description> can be presented as follows:

```
<description> ::= <classID>
                | <restriction>
                | unionOf( {<description>} )
                | intersectionOf( {<description>} )

                | complementOf( <description> )

                | oneOf({<individualID>} )
```

(4) Summary of DIM OWL Implementation

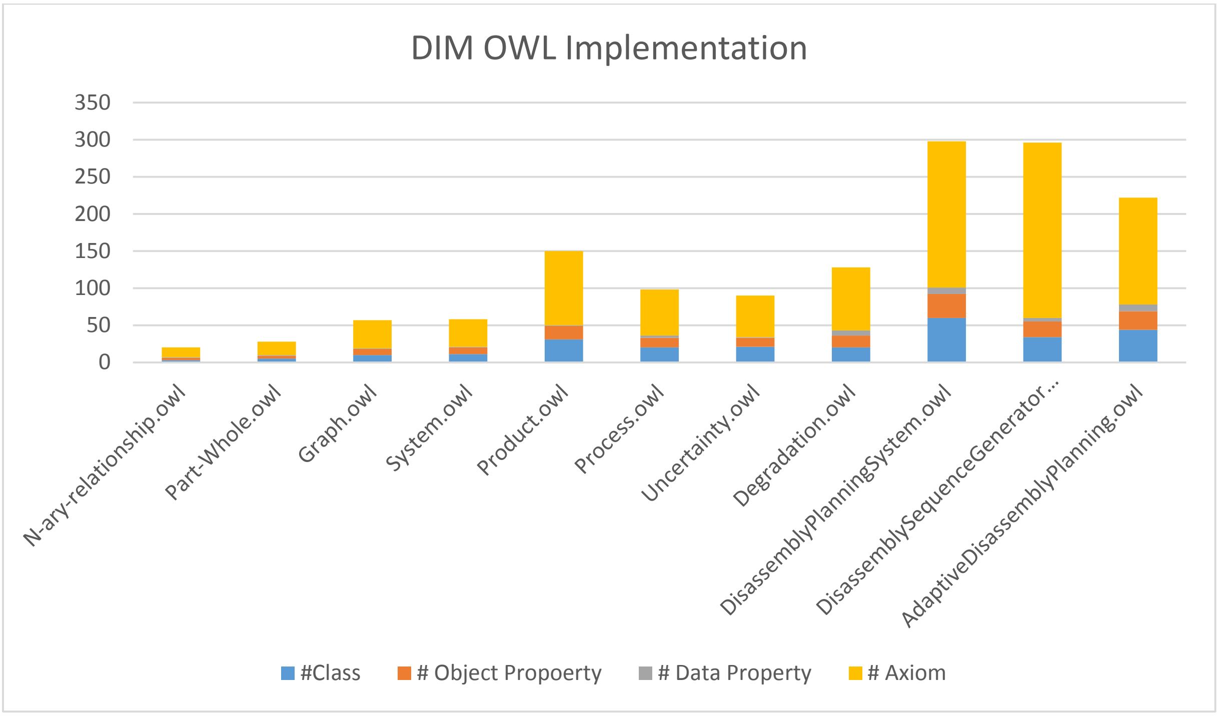


Figure 3.26: DIM OWL Implementation Summary

Figure 3.26 above summarizes the DIM OWL implementation according to four schema metrics (we include the two application level sub models as well, which are presented in chapter 4&5): (1) number of classes, (2) number of object properties, (3) number of data properties and (4) number of axioms. The detailed descriptions of the major DIM OWL implementation are presented in the appendix.

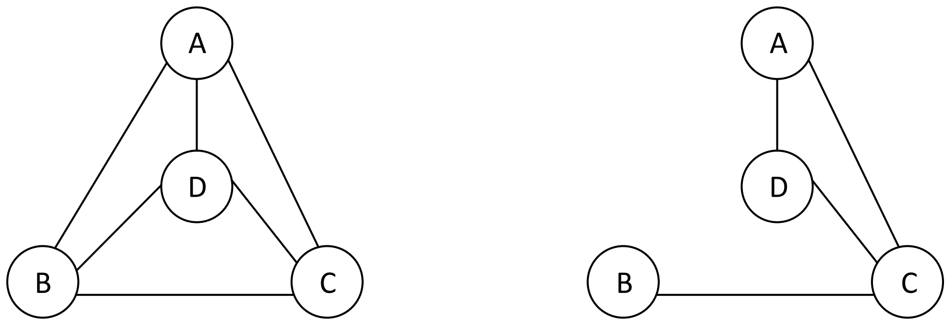
Chapter 4 DISASSEMBLY SEQUENCE GENERATOR

In this chapter 4 and next chapter 5, two disassembly planning applications are developed and presented, with the intension to validate the reusability and usability quality of the proposed DIM. Chapter four focuses on the problem of the disassembly sequence generation, which targets at finding all the feasible disassembly sequences of an EOL product and further locating the economically optimized one. We start with the disassembly sequence generation problem definition and background review in section 4.1. Next, the Disassembly Sequence Generator Information Model, residing on the application layer of the developed DIM, is presented in detail in section 4.2. Section 4.3 presents an example to populate the proposed IM. The detailed application algorithm for carrying out the sequence generation and optimization process in presented in section 4.4. Lastly, the chapter is wrapped up in section 4.5 with a case study to verify the application procedure presented in section 4.4.

4.1 Disassembly Sequencing Problem

Disassembly sequences are listings of disassembly processes (such as the separation of an assembly into two or more subassemblies, or removing one or more connections between components), through which an EOL product can be separated into small pieces. Unlike the assembly process, which usually follows a pre-defined sequence of steps to achieve the final deliverable, most of the disassembly planning yields multiple feasible disassembly sequences. A disassembly sequence is said to be feasible if it satisfies the geometrical and topological constraints related to the EOL product. Detailed definitions of these constraints are described below:

Topological Feasibility: topological feasibility is related to the connections in an EOL product. In an ideal, so called "strongly connected product" case, where each component in the product is connected with all the other components (contains the maximum number of possible connections), every subset of the components can be considered as a topologically feasible subassembly (no topological constraints are present). As an example shown in figure 4.1 (a) below: a product with four components is being classified as a strongly connected product because each component in the product is connected with all the other components (like component A is connected to component B, C and D). Thus, all the combinations of the components can be considered as a feasible subassembly (AB, AC, ABC, etc.).



(a) Strongly Connected Product Example (b) Weakly Connected Product Example

Figure 4.1: Topological Feasibility Examples

However, in a real life situation, the number of connections maybe far less than that in the maximum possible case, which imposes certain topological constraints onto the product. In a weakly connected product, there always exists at least one subset of components that is not connected. Therefore, that subset does not correspond to a subassembly. Figure 4.1 (b) shows a non-strongly connected product scenario. In this example, subset AB is not a subassembly because component B is not connected to component A. Such topological constraint yields the infeasibility of detaching component A and component B together as a subassembly.

<u>Geometrical Feasibility</u>: geometrical constraints explains the impracticality of specific disassembly processes which are obstructed geometrically by the presence of some other components. Two levels of geometrical constraints are necessary to be considered in order to define the feasibility of a certain disassembly process:

- Detachability: the ability related to whether a component or subassembly can be detached without interference (i.e. A collision free path exists for the detachment to take place).

- Stability: the ability of a product to hold its components together in a stable manner. A feasible disassembly process should not yield an unstable subassembly (where the subassembly falls apart spontaneously) without any further disassembly process. As an example, in figure 4.2, both of the two examples are stable initially. A disassembly process: “the detachment of part C” is under study, which will yield an unstable subassembly (part B is movable) in both examples. However, “the detachment of part C” in example 2 can be followed by another feasible disassembly process (detachment of part B) and finally result in the full disassembly of the product. Thus, we still consider “the detachment of part C” a feasible disassembly process for the product in example 2, even though it results in an unstable state. However, in example 1, part B is not detachable from the product after part C is detached (no further sequential feasible disassembly process exists). Thus, “the detachment of part C” is not a feasible disassembly process for example 1 in figure 4.2.

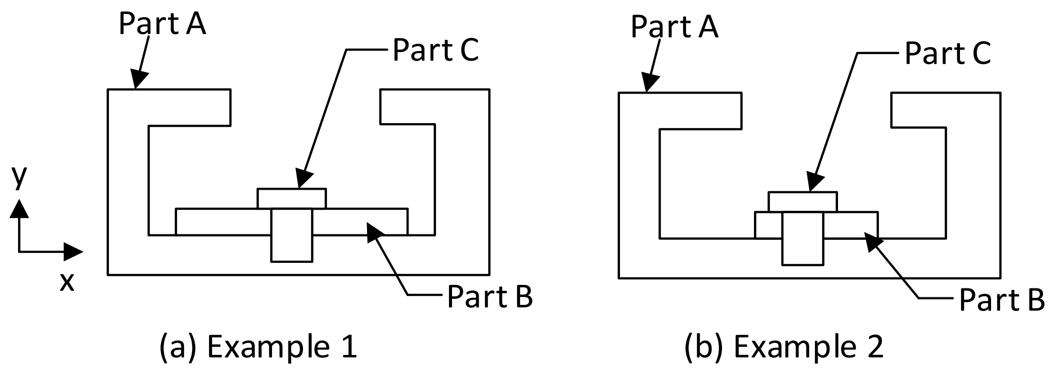


Figure 4.2: An Example to Explain the Product Stability

Locating all the feasible disassembly process sequences is only the first objective of the disassembly sequencing problem; the second one is to use an optimization technique on all the feasible disassembly sequences for obtaining the economically optimal disassembly process. The objective of the optimization model is to find a best “disassembly path” (among the feasible set of disassembly sequences), which achieve a minimized disassembly process cost and maximized retrieved components' revenue. The details of the optimization model are discussed in section 4.4.

In summary, the objectives of the disassembly sequencing problem can be categorized into two sub problems:

1. Identify all feasible disassembly process sequences
2. Obtain the optimal disassembly process sequence considering economic benefits

4.2 Disassembly Sequence Generator Information Model

The details of the Disassembly Sequence Generator Information Model are presented in this section. We start with the information requirement analysis in section 4.2.1 and the formal model is presented in section 4.2.2 using UML class diagram, with the OWL implementation summarized at the end.

4.2.1 Disassembly Sequence Generator Information Requirement Analysis

From the problem definition presented in the section 4-1, the information required for solving the disassembly sequencing problem can be considered from three aspects as follows:

- *Information related to the product's topological configuration*. In the Product Model presented in chapter 3, information related to the EOL product structure or topology has been modeled by introducing the classes **Product**, **SubAssembly** and **Component** (refer to figure 3.13 for details). In the disassembly process, more detailed classification of the **SubAssembly** class should be elaborated. As an example, in figure 4.3, "Part6-Part1-Part2" can be an instance of the **SubAssembly** class, since they are topologically connected (Part 6 is connected with Part 1 and Part 1 is connected to (contact connection) Part 2). However, in the view of EOL product disassembly, such combination is not realistic. We would rather pick subassembly "Part6-

Part1-Part5" or subassembly "Part2-Part8-Part3-Part9-Part4" for candidate subassemblies to be detached from the EOL product. Thus, two new types of subassembly, called **ContactLoop** and **ContactLoopCluster**, are modeled to better serve the disassembly sequencing problem and their formal definitions are given in detail in section 4.2.2.

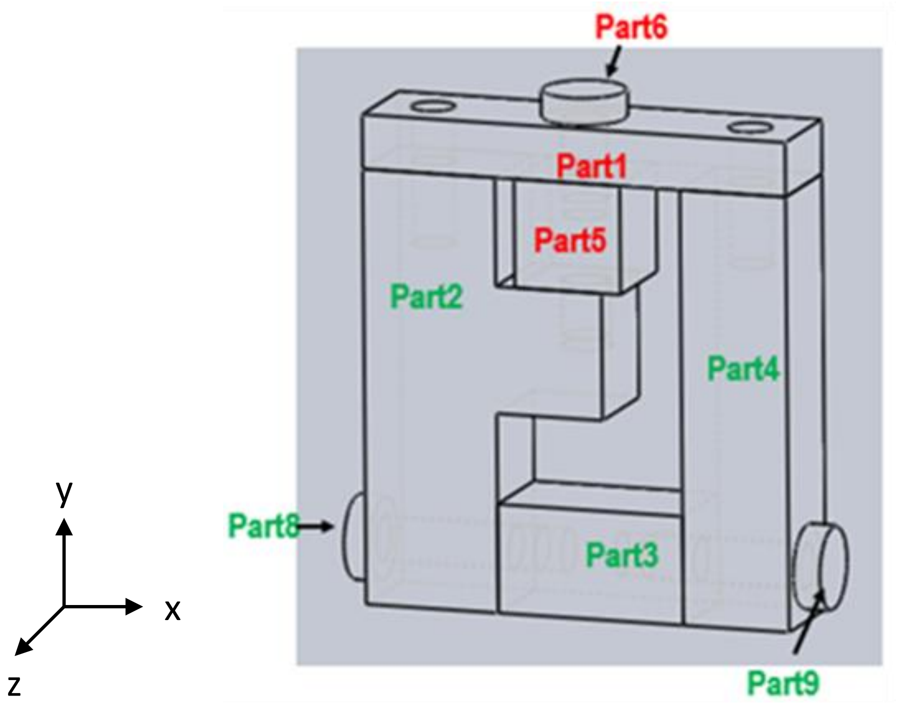


Figure 4.3: An Example to Explain Product Topological Configuration

- *<u>Information related to the product's geometrical constraints.</u>* Geometrical constraints are the most important considerations in the planning of disassembly, which usually can be further broken down into two types: the local geometrical constraints and the global geometrical constraints. The local geometrical constraints restrict the components from moving along certain directions, whereas the global geometrical constraints restrict the component from being fully detached from the EOL product. Let us take the product from figure 4.2 (a) as an example, Part C is locally constrained by Part B and Part A along the ±x direction and –y direction and is globally constrained by Part A along the ±x direction. However, Part C is detachable along + y direction, thus there is no global geometrical constraints along that direction. The modeling of the global geometrical constraints requires the full descriptions of the boundary representation of the whole product, which will yield a very large information structure. We thus only include the information elements related to the local geometrical constraints in the Disassembly Sequence Generator Information Model, the global geometrical constraints are being handled using a CAD-API based simulation approach. This way, the complex boundary representation of the whole product is condensed into one piece of information which indicates the location and name of the related CAD file. The details of the simulation approach are elaborated in section 4-4.

- *Economic Information.* Last information requirement relates to the economic evaluation of the disassembly plan. The evaluation is based on the revenue that disassembly operators can expect from the retrieved component or subassembly and the cost being spent through carrying out the disassembly process. Such information is needed for the disassembly optimization process.

4.2.2 Formal Disassembly Sequence Generator Information Model

The Disassembly Sequence Generator Information Model deals with the information required for the disassembly sequencing problem. It is residing on the application layer of the DIM and is being extended based on the domain layer Product Model. The overall structure is shown in figure 4.4 below. We will describe the model according to the information requirements identified above in the following sections.

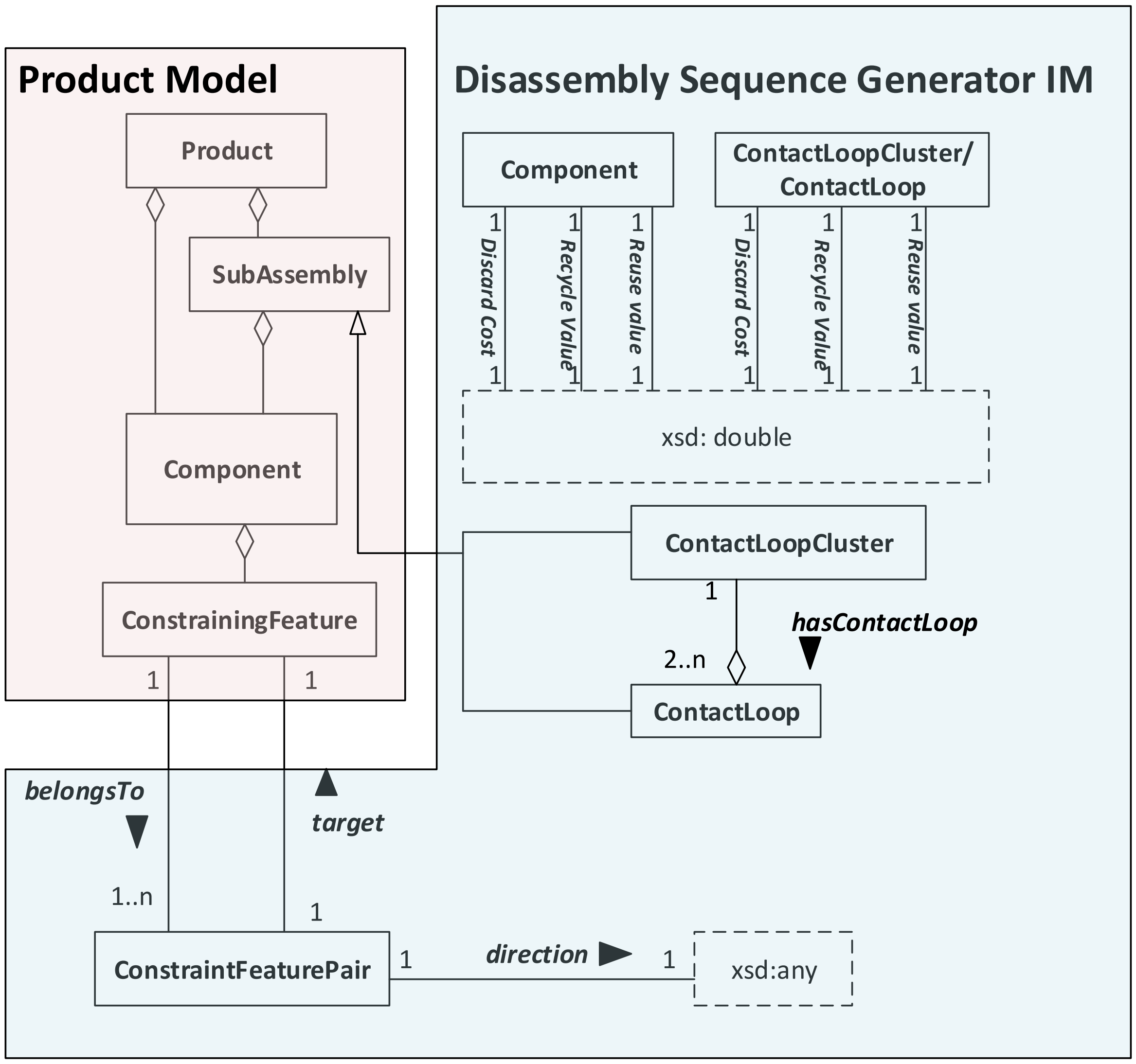


Figure 4.4: Structure of the Disassembly Sequence Generator Information Model

R1: *Information related to the product topological configuration*

As mentioned above, two special types of the **SubAssembly** class, named **ContactLoop** and **ContactLoopCluster** are introduced for the disassembly sequencing problem. The details of these two concepts are presented below:

<u>Concept of ContactLoop</u>

The main idea behind the **ContactLoop** concept is that most of the mechanical connections involves a set of components that together forms a loop in the corresponding product connection diagram. Figure 4.5 explains the concept with examples. The top left example is a simple screw connection which connects two **OrdinaryComponents** (Part A and Part B) using a screw (**ConnectingComponent** Part C). In its connection diagram, there exists a loop among Part A, Part B and Part C (Part A has a contact connection with Part B, Part B has a threaded connection with Part C and Part C has a threaded connection with Part A). Similar observation can be found in the top right example where a screw is used to connect more than two components (Loop "Part A->Part B ->Part C", loop "Part B->Part D->Part C" and loop "Part A->Part B->Part D->Part C").

We call such loop **ContactLoop**, which is a special type of the **SubAssembly** class and forms a "*<u>building block</u>*" for various complex mechanical connections: Different types of complex connection are an aggregation of **ContactLoops**, we will explain more when describing the concept of **ContactLoopCluster** in next section.

The concept of **ContactLoop** plays a critical role in the disassembly sequencing analysis: in every stage of the disassembly planning, we need to identify such a subassembly so that we can efficiently detach a set of components together (parallel disassembly) instead of only detaching one component from the whole product (sequential disassembly).

Formally, for all the loops in an EOL product connection graph, if the loop has the following properties, it is a **ContactLoop**:

ContactLoop is a loop, in which

- Only one **ConnectingComponent** exists in the Loop.
- All the **OrdinaryComponent** are constrained by the **ConnectingComponent** in the loop, (i.e. all the **OrdinaryComponents** are connected to or have contact with the **ConnectingComponent** in the loop).

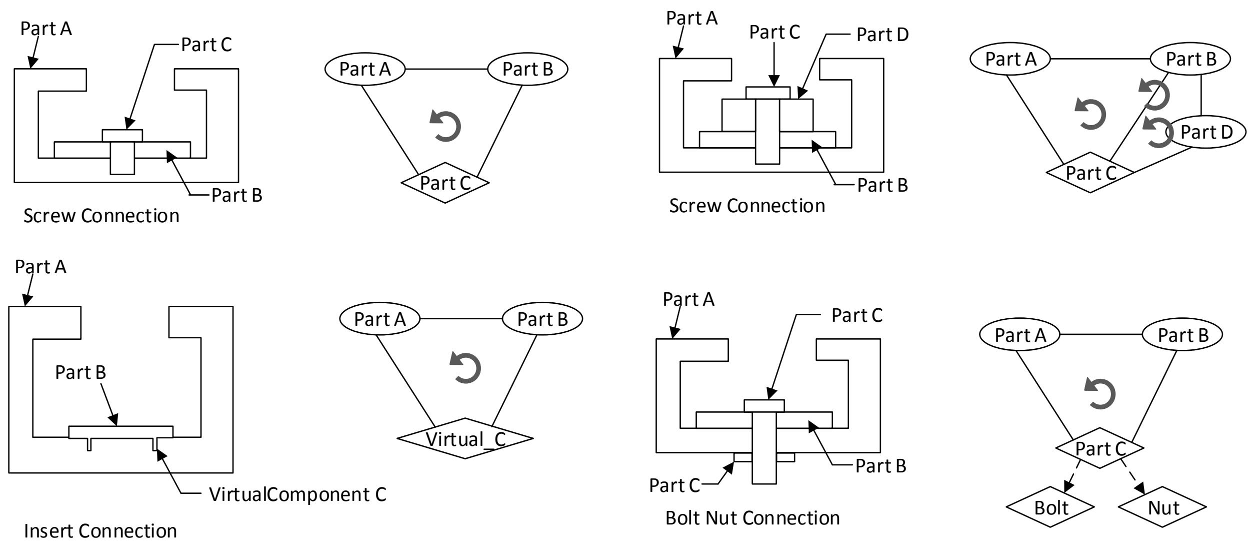


Figure 4.5: Examples of the ContactLoop Concept

The concept of the **ContactLoop** is not restricted to the screw connection only, it can be well applied to the other types of connections with minor modifications. For example, for the insert connection (example on the bottom left of figure 4.5), where there is no **ConnectingComponent** involved, the concept of **VirtualConnectingComponent** introduced in chapter 3 can be used to mimic the role of the **ConnectingComponent.** For the Bolt-Nut connection (example on the bottom right of figure 4.5), two **ConnectingComponents** (bolt and nut) are involved. However, the connecting function is based on the joint effort of the bolt and the nut. Either one alone cannot provide the connection function and thus cannot be considered as a **ConnectingComponent**. Also, from the disassembly point of view, almost always bolt and nut are detached sequentially together. Thus, we treat bolt and nut together as one **ConnectingComponent.** Under such mechanism, a contact loop will be identified as well.

Concept of ContactLoopCluster

If we cluster a set of **ContactLoops**, more complex subassembly will be created and we call such subassembly **ContactLoopCluster**. Formally, the definition of **ContactLoopCluster** is:

ContactLoopCluster:

A Combination of **ContactLoops,** among which one or more **OrdinaryComponents** are being shared by two or more **ContactLoops**.

Figure 4.6 gives an example of the **ContactLoopCluster** concept. As shown in the contact diagram, Part 4, Part 9 and Part 3 forms a **ContactLoop** and similarly, Part 2, Part 8 and Part 3 forms another **ContactLoop.** Both of the loops share the same **OrdinaryComponent** (Part 3), thus "Part 4, Part 9, Part 3, Part 8 and Part 2" forms a **ContactLoopCluster.** It is evident from figure 4.6 that the identified **ContactLoopCluster** forms a more complex subassembly, compared to the original **ContactLoops**.

In some cases, a **ContactLoop** can itself be a **ContactLoopCluster.** In the top right example in figure 4.5, three **ContactLoops** are identified:

L1: Part A->Part B ->Part C (Part C is **ConnectingComponent**)

L2: Part B->Part D->Part C (Part C is **ConnectingComponent**)

L3: Part A->Part B->Part D-><u>Part C </u>(Part C is **ConnectingComponent**)

Among the above three **ContactLoops**, L3 is also a **ContactLoopCluster** since it is a combination of L1 and L2 by sharing the same **OrdinaryComponent** Part B.

Similar to the reason for introducing the concept of **ContactLoop** into disassembly sequencing, identifying the **ContactLoopCluster** first will result in a more efficient disassembly process (parallel disassembly).

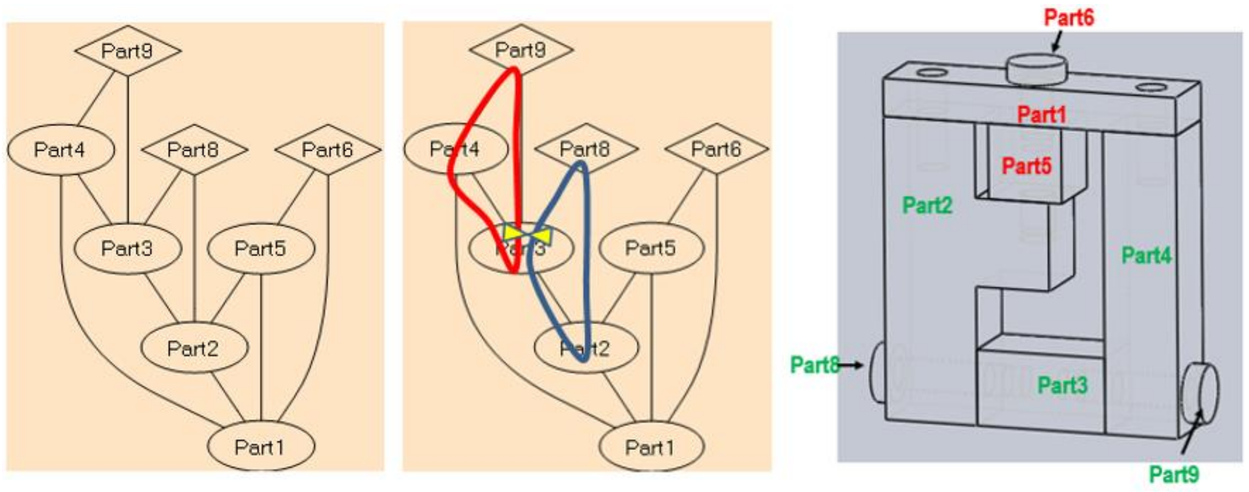


Figure 4.6: An Example of the ContactLoopCluster Concept

R2: *<u>Information related to the product local geometrical constraints</u>*

The local geometrical constraints are modeled by extending the class **ConstrainingFeature** located in the Product Model. Recall, the class **ConstrainingFeature** represents the interface feature through which a component is connected to (or constrained by) another component. However, in the original Product Model, how the component is constrained by the **ConstrainingFeatures** of the connecting components is unknown (we can only know what **ConstrainingFeature** a component has). In other words, we need to combine pair wise **ConstrainingFeatures** of two connected components. The class **ConstraintFeaturePair** is developed for such purpose, which represents a placeholder to relate two **ConstrainingFeatures** involved in a connection, by introducing the object properties ***belongsTo*** and ***target***. Also, the **ConstrainingFeature** of a component restrains the component from being detached along a certain direction. Such information is modeled by the data property "*direction*" attached to the **ConstraintFeaturePair** class.

The example in figure 4.7 is used to explain the above concepts. Let's look at the local constraints of *Component A*: it is being locally constrained by *Component C* along +X and –X direction and being locally constrained by *Component B* along –Y direction.

When mapping the above information to the Information Model concepts discussed above, we first can know *Component A* has two **ConstrainingFeatures** (*A-f1* and *A-f2*), through which it is being locally constrained. Both *A-f1* and *A-f2* belong to a **ConstraintFeaturePair** instance (*ConstrainedFeaturePair_1* and *ConstrainedFeaturePair_2* respectively), which can be identified by the ***belongsTo*** object property.

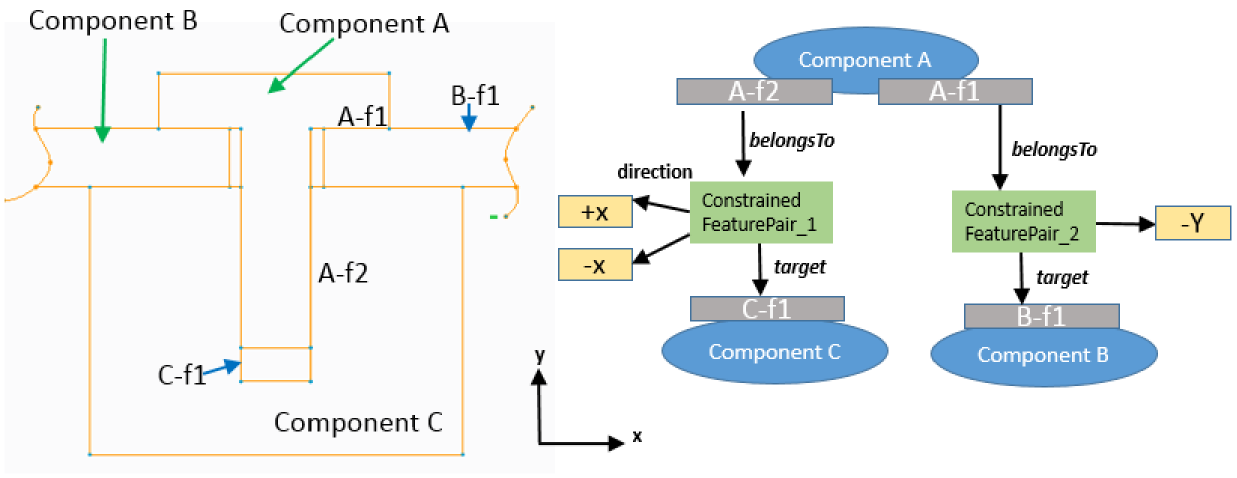


Figure 4.7: An Example of Modelling the Local Geometrical Constraints

After locating the **ConstraintFeaturePair** information, we can further know the **ConstrainingFeature** information of the other component from which Component A is being constrained, through the object property ***target***. Take *ConstrainedFeaturePair_1* as an example, we can know that **ConstrainingFeatures** *A-f1* from *Component A* is constrained by the **ConstrainingFeatures** *C-f1* from *Component C*. We can also know, *C-f1* is constraining *Component A* along the *+X* and *–X* directions.

R3: *Economic Information*

Economic information can be separated into two aspects:

- Related to the disassembly object. This includes the reuse value, recycle value and discard cost of the **Component** and the **ContactLoopCluster**.
- Related to the disassembly process. This includes the average process cost and special process cost and they have been modeled in the Process Model in chapter 3.

The related information modeling elements are shown in figure 4.4: 3 data properties (ReuseValue, RecycleValue and DiscardCost) have been included to present economic information related to the **Component** class and the **ContactLoopCluster** class.

4.2.3 OWL implementation

The above Disassembly Sequence Generator Information Model has been implemented in OWL by extending the Product Model residing on the domain layer of DIM and the relevant concepts have been summarized in table 4.1 below.

4.3 Populating the DIM

In this section, we will use the model shown in figure 4.8 as an illustrative example for populating the Disassembly Sequence Generator Information Model. This exemplary model is used throughout this chapter for illustration and verification purposes. Relevant information in the case study problem is populated into the related classes (**Component**, **ConstrainingFeature** and **ConstrainingFeaturePair)** and figure 4.9 shows a screenshot of the Protégé implementation. We expand the content in figure 4.9 and show only the detailed information related to one of the component (*Part1*) in figure 4.10 (Information related to the other components are identical.)

Table 4.1: Summary of the DIM OWL Implementation Concept

Model	Imported Model	Class	Class Axioms	Object Property	Datatype Property
DisassemblySequence Generator.owl	Product.owl	Constraining FeaturePair	target **exactly** 1 ConstrainingFeature	target	reuseValue
			direction **exactly** 1 string	belongsTo	recycleValue
		Component	discardCost **exactly** 1 double	hasContact Loop	discardCost
			recycleValue **exactly** 1 double		direction
			reuseValue **exactly** 1 double		
		Constraining Feature	belongsTo **some** ConstrainingFeaturePair		
		ContactLoop	Subclass of SubAssembly		
			discardCost **exactly** 1 double		
			recycleValue **exactly** 1 double		
			reuseValue **exactly** 1 double		
		ContactLoop Cluster	Subclass of SubAssembly		
			discardCost **exactly** 1 double		
			recycleValue **exactly** 1 double		
			reuseValue **exactly** 1 double		
			hasContactLoop **min** 2 ContactLoop		

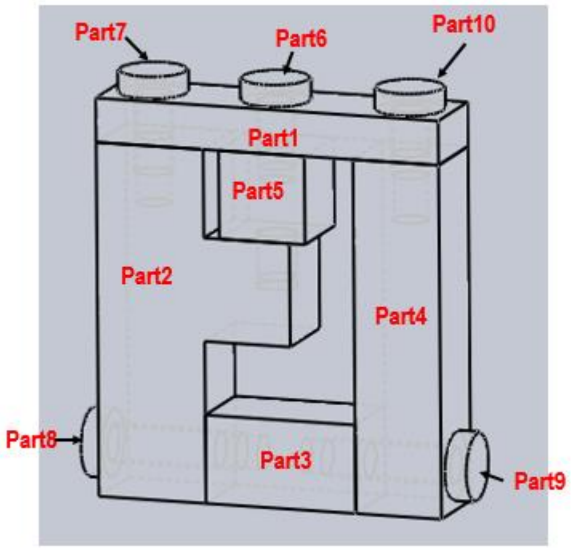
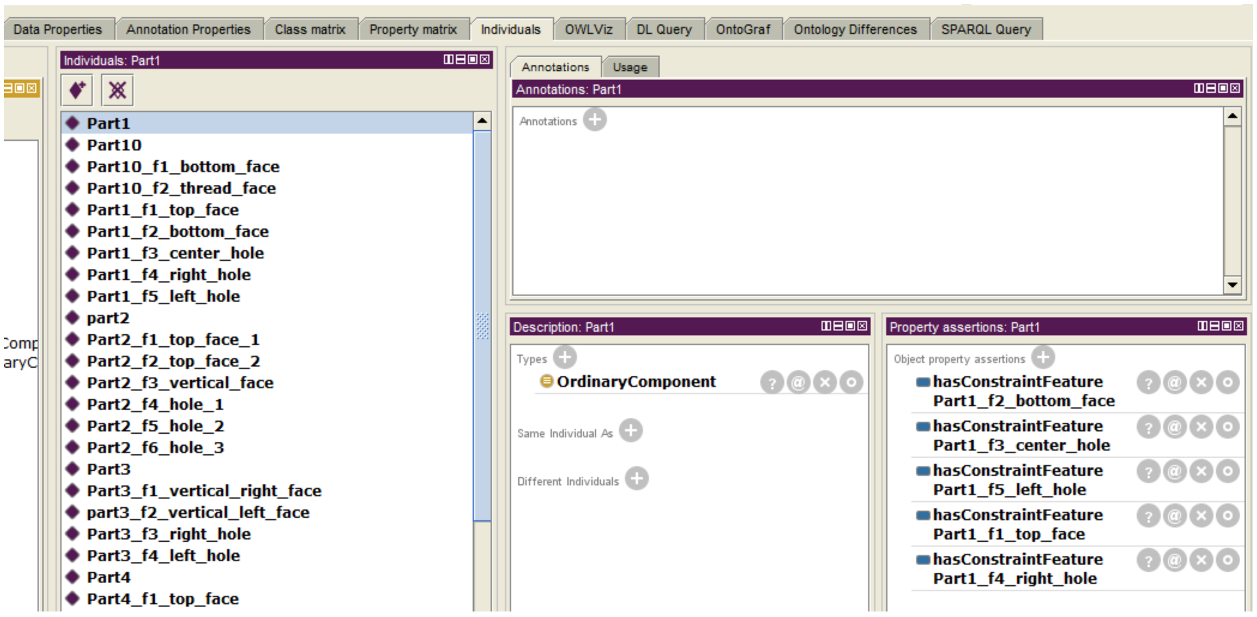


Figure 4.8: An Illustrative Example

Since *Part 1* is not functioning as connecting purpose, it is being classified as an instance of the class **OridinaryComponent** and it has five **ConstrainingFeatures** as follows:

Part1_f1_top_face: top face of part 1 (feature #1)
Part1_f2_bottom_face: bottom face of part 1 (feature #2)
Part1_f3_center_hole: hole feature in the center of part 1 (feature #3)
Part1_f4_right_hole: hole feature in the right of part 1 (feature #4)
Part1_f5_left_hole: hole feature in the left of part 1 (feature #5)



Individuals: Part1

- Part1
- Part10
- Part10_f1_bottom_face
- Part10_f2_thread_face
- Part1_f1_top_face
- Part1_f2_bottom_face
- Part1_f3_center_hole
- Part1_f4_right_hole
- Part1_f5_left_hole
- part2
- Part2_f1_top_face_1
- Part2_f2_top_face_2
- Part2_f3_vertical_face
- Part2_f4_hole_1
- Part2_f5_hole_2
- Part2_f6_hole_3
- Part3
- Part3_f1_vertical_right_face
- part3_f2_vertical_left_face
- Part3_f3_right_hole
- Part3_f4_left_hole
- Part4
- Part4_f1_top_face

Comp
aryC

Annotations: Part1

Annotations

Description: Part1

Types
OrdinaryComponent

Same Individual As

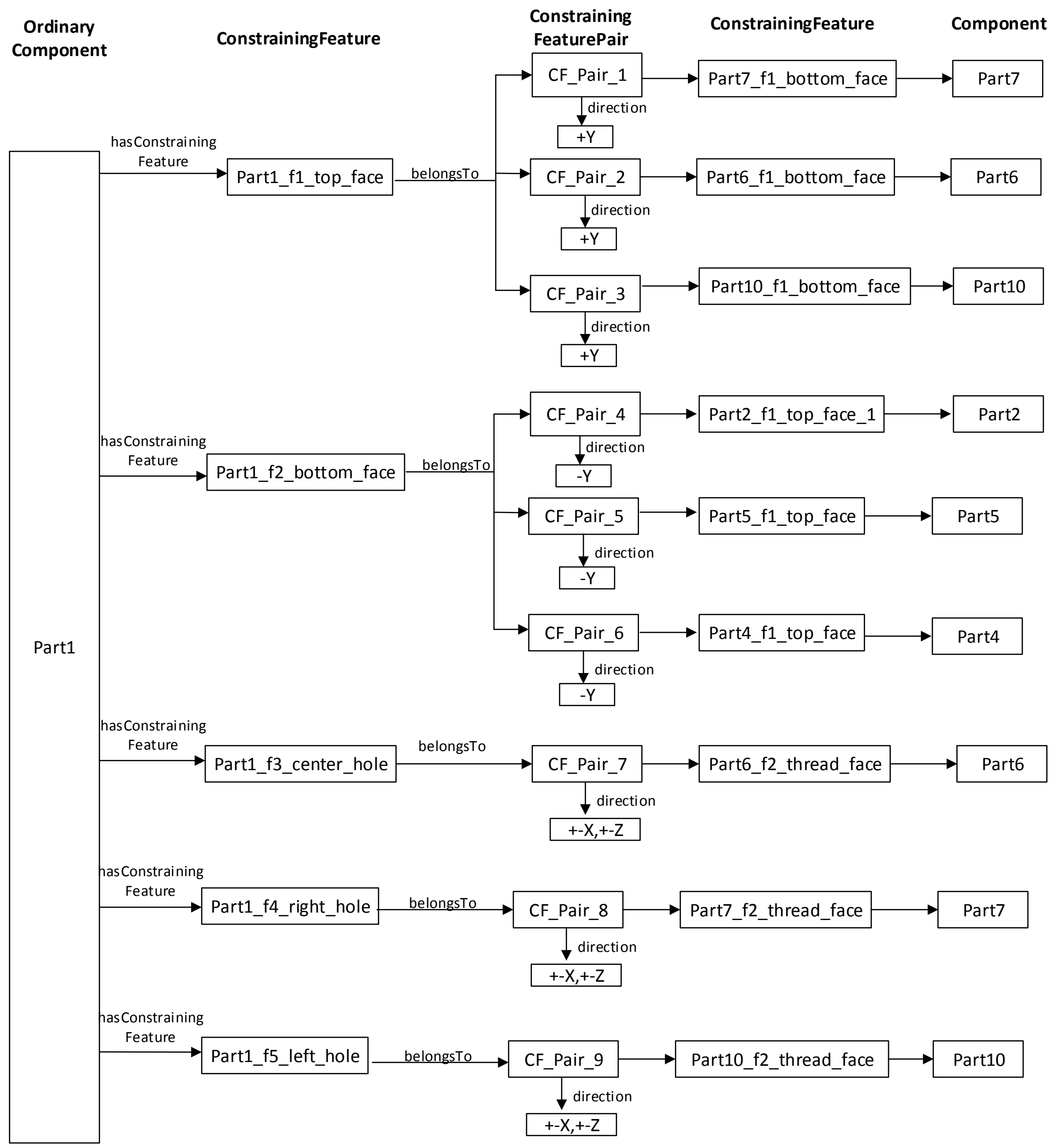
Different Individuals

Property assertions: Part1

Object property assertions
hasConstraintFeature Part1_f2_bottom_face
hasConstraintFeature Part1_f3_center_hole
hasConstraintFeature Part1_f5_left_hole
hasConstraintFeature Part1_f1_top_face
hasConstraintFeature Part1_f4_right_hole

Figure 4.9: Instance Population in Protégé

Each **ConstrainingFeature** instance belongs to a certain **ConstrainingFeaturePair,** through which the **ConstrainingFeature** information of the connected component can be revealed. As an example, *CF_Pair_1* is one of the **ConstrainingFeaturePair** instance and it relates one of the **ConstrainingFeature** of Part 1(*Part1_f1_top_face*) to its connected component *Part 7*, through the **ConstrainingFeature** of *Part 7* (*Part7_f1_bottom_face*). Also, we can know, a direction information is attached to the **ConstrainingFeaturePair** *CF_Pair_1*, which indicates that *Part 1* is being locally constrained by *Part 7*'s bottom face feature (*Part7_f1_bottom_face*) along +Y direction.



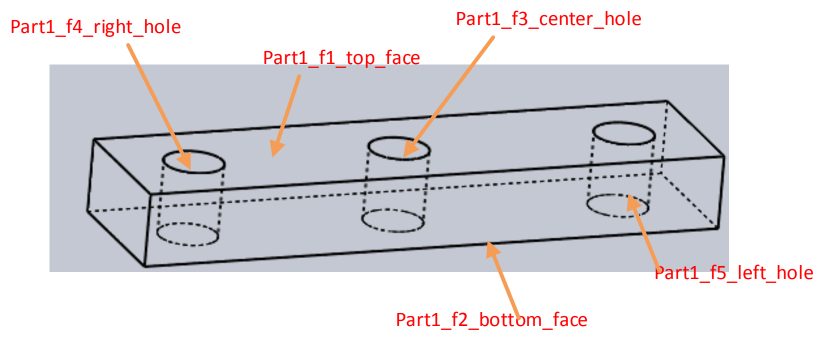
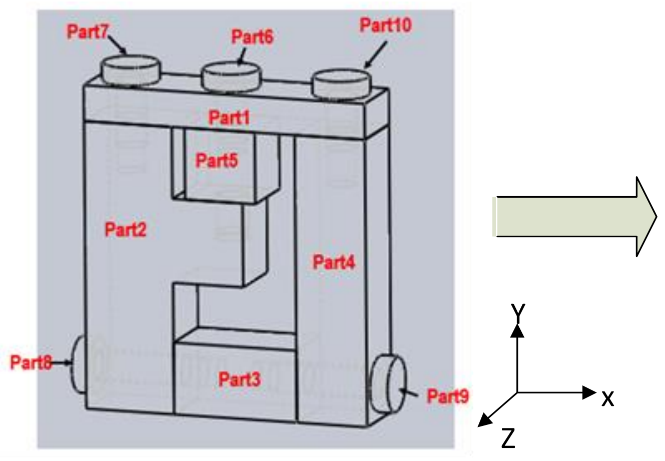
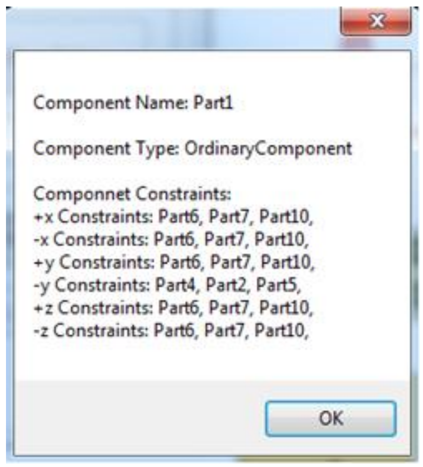


Figure 4.10: Detailed Populated Information about Part1

In the disassembly planning, one important information is to locate the local constraints of a component and such information is explicitly represented in the proposed DIM (refer to the example in figure 4.10). A simple API call is developed to collect the information in a more organized way. The exemplary output is shown in figure 4.11, which presents the local constraints of *Part1*, along 6 principle axis.





```
Component Name: Part1

Component Type: OrdinaryComponent

Componnet Constraints:
+x Constraints: Part6, Part7, Part10,
-x Constraints: Part6, Part7, Part10,
+y Constraints: Part6, Part7, Part10,
-y Constraints: Part4, Part2, Part5,
+z Constraints: Part6, Part7, Part10,
-z Constraints: Part6, Part7, Part10,
```

Figure 4.11: Local Constraints of Part 1

4.4 A CAD API based Disassembly Sequence Generation Application

The overall structure of the Disassembly Sequence Generation application is presented in figure 4.12 below. The inputs to the application are the OWL implementation file for the Disassembly Sequence Generator Information Model and the product CAD file. The OWL file contains the necessary information structure for the sequencing problem and the product CAD file is used here to handle the component global constraints, which is not included in the Disassembly Information Model. The output of the application is a theoretically optimal disassembly process sequence, based on the geometrical, topological and economic considerations. We consider this result theoretic due to the fact that no disturbances or uncertainties are considered in this application and it is optimal only if the status of all the components is like new and all the processes can success without failure. A more realistic application which addresses the uncertainty issues is presented in chapter 5.

The Disassembly Sequence Generator application can be broken into two parts: (1) the disassembly sequencing, which focuses on identifying all the feasible disassembly process sequences of an EOL product and (2) the Linear Programming (LP) based optimization, which takes the result (AND/OR graph) from the first part as the input and find the economically optimal process sequence. The disassembly sequencing part further consists of three main tasks: (1.1) Construct "EOLProduct" object, (1.2) Interference Test and (1.3) Unconstrained Subassembly Detection. Each of the tasks is presented in detail in the following sections.

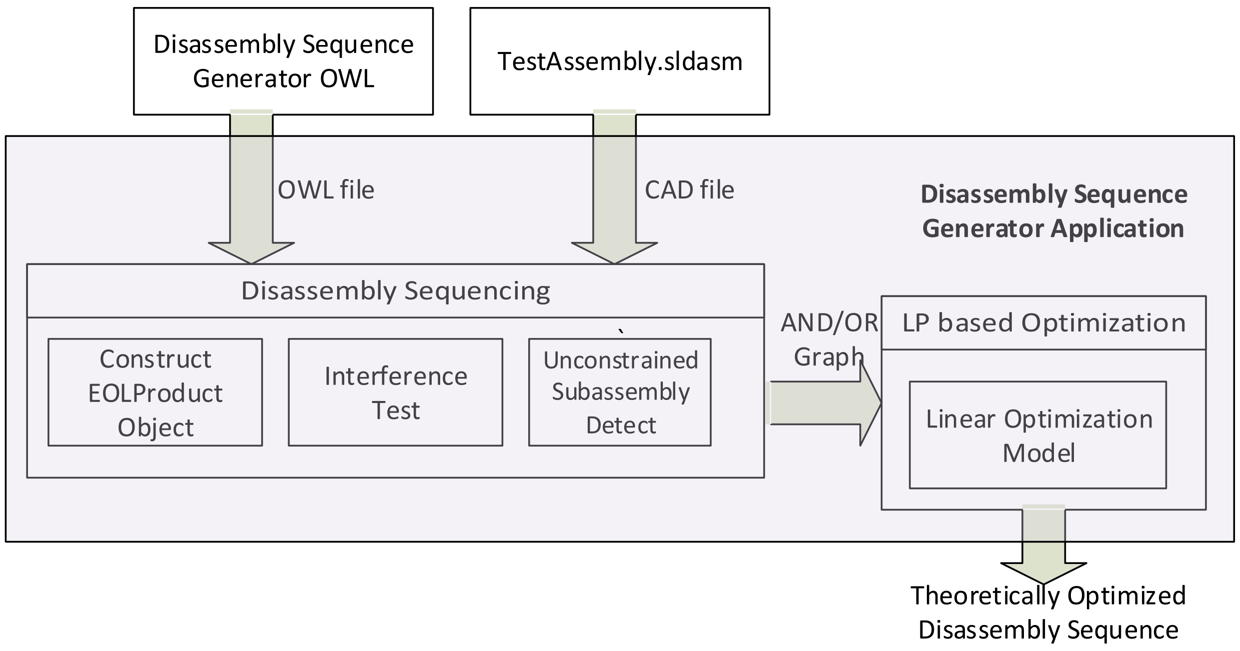

Disassembly Sequence Generator OWL
TestAssembly.sldasm
OWL file
CAD file
Disassembly Sequence Generator Application
Disassembly Sequencing
Construct EOLProduct Object
Interference Test
Unconstrained Subassembly Detect
AND/OR Graph
LP based Optimization
Linear Optimization Model
Theoretically Optimized Disassembly Sequence

Figure 4.12: The Overall Structure of the Disassembly Sequence Generator Application

4.4.1 Disassembly Sequencing

This section presents the details of the first part of the Disassembly Sequence Generator application, which targets on identifying all the feasible disassembly process sequences of an EOL

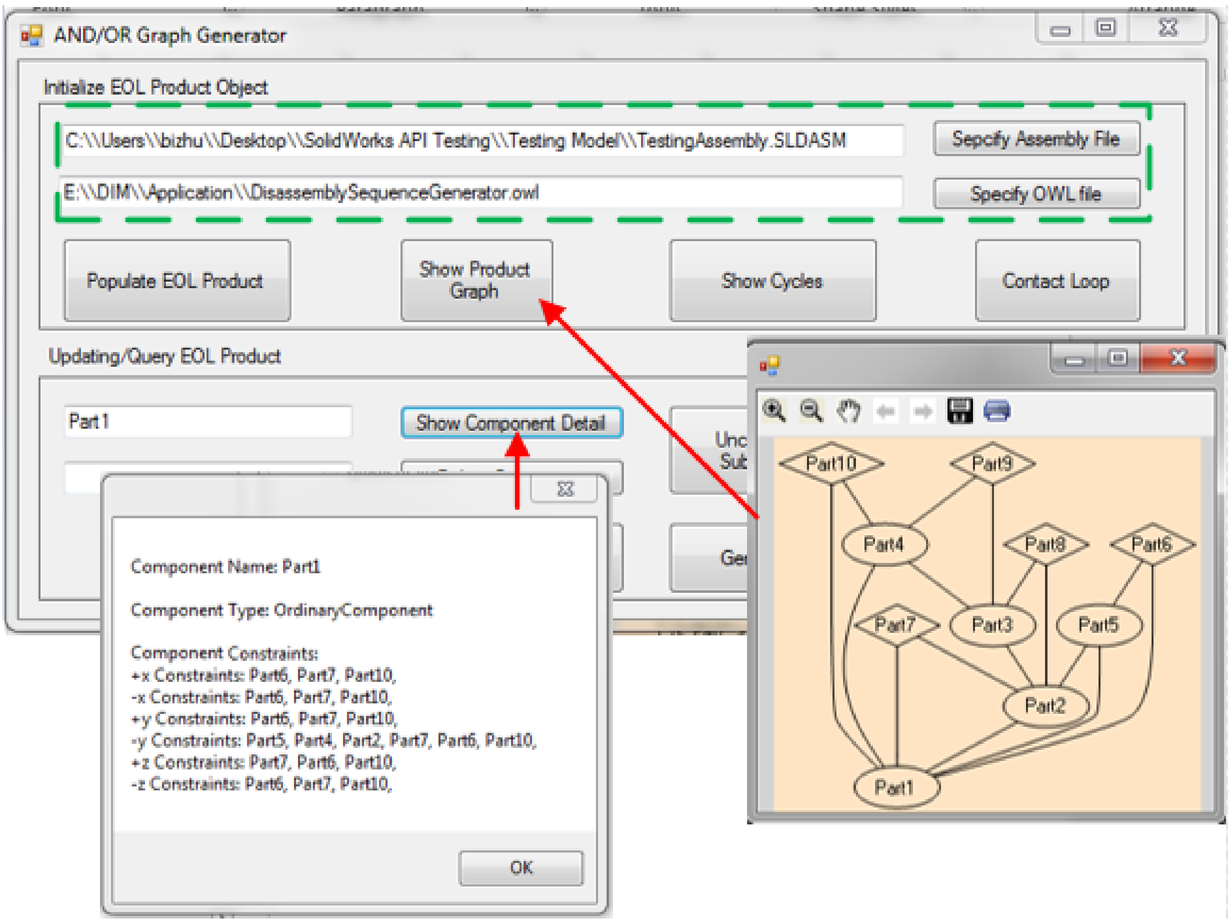
Product. Three major involved tasks are elaborated first in the following paragraphs and then the overall application procedure is presented at the end of section 4.4.1.

Construct the "EOLProduct" Object

The first task is to extract information from the OWL file and organize them into a certain programming object, called "EOLProduct", so that the application program can process the information easily. In other words, this is the information preparation stage of the whole application. The structure of the "EOLProduct" object is presented as follows:

```
public class EOLProduct
{
    public List<Component> ordinaryComponentList { get; set; }
    public List<Component> connectingComponentList { get; set; }
    public List<Component> allComponentList { get; set; }
    public Graph connectionGraph { get; set; }
    public String file { get; set; }
}
public class Component
{
    public String name { get; set; }
    public ComponentType type { get; set; }
    public String[] positiveXConstraints { get; set; }
    public String[] negativeXConstraints { get; set; }
    public String[] positiveYConstraints { get; set; }
    public String[] negativeYConstraints { get; set; }
    public String[] positiveZConstraints { get; set; }
    public String[] negativeZConstraints { get; set; }
    public String associatedAssemblyFile { get; set; }
}
```

The "EOLProduct" object contains three lists ("ordinaryComponentList", "connectingComponentList" and "allComponentList") registering the different types of components in an EOL product (**OrdinaryComponent**, **ConnectingComponent** and general **Component**). Each component is further an aggregated object, which comprises of the name information, the component type information and the local constraint information. In this thesis, we only consider 6 primary Cartesian directions (±X, ±Y and ±Z) as the possible disassembly directions. Thus, the "local constraint" information is recorded in an array, which contains the components that are geometrically restricting the current component moving along a certain primary Cartesian direction. Also, a “file” information is included to record the associated CAD file name of the EOL Product. Lastly, all the components are organized into a “connectionGraph” object, which represents the topological arrangements of the components.



C:\\Users\\bizhu\\Desktop\\SolidWorks API Testing\\Testing Model\\TestingAssembly.SLDASM

E:\\DIM\\Application\\DisassemblySequenceGenerator.owl

Part1

Component Name: Part1

Component Type: OrdinaryComponent

Component Constraints:
+x Constraints: Part6, Part7, Part10,
-x Constraints: Part6, Part7, Part10,
+y Constraints: Part6, Part7, Part10,
-y Constraints: Part5, Part4, Part2, Part7, Part6, Part10,
+z Constraints: Part7, Part6, Part10,
-z Constraints: Part6, Part7, Part10,

Figure 4.13: Organizing Information into the “EOLProduct” Object

Figure 4.13 shows the implementation of the Disassembly Sequence Generator application, with annotations on the elements relating to the “Construct EOLProduct Object” task. The inputs of the application are two files: (1) the Disassembly Sequence Generator OWL file and (2) the EOL product CAD file and the locations of both files are being specified by the user. After locating the two inputs, the OWL file will be queried and the retrieved information will be used to construct the “EOLProduct” object. The result can be validated by checking the product connection graph or the detailed component information, through button “Show Product Graph” and the button “Show Component Detail”.

As an example, the detail information of Part1 is shown in lower dialog box in figure 4.13. It contains the name of the component (Part1), the type of the component (**OrdinaryComponent**), and the local component constraints along six primary directions (e.g. Part1 is being locally constrained by Part6, Part7 and Part 10 along +X direction).

Interference Test

The goal of the interference test task is to address the issue related to the global constraints of the component and to check whether a component can be detached from the product along a primary direction without collisions with any other components. Since the detailed geometrical form information is not modeled in the Disassembly Information Model, the interference test task utilizes a CAD simulation based approach to check the detachability of the component. The detail procedure is a recursive process as shown in figure 4.14 (we take the interference test function along positive X direction as an example).

The function starts with the initialization of the CAD programming objects related to the product under study, and Solidworks is used in this thesis for the implementation. There are three main SolidWorks document types, namely Part Document, Assembly Document and Drawing Document and each document type has its own programming object (PartDoc, DrawingDoc and AssemblyDoc), through which the user can manipulate the CAD model programmatically. In the proposed procedure, the “swApp” object is used to start the Solidworks application and the

"AssemblyDoc" object is used to provide access to the functions that perform certain assembly operations.

The second step is to identify the size of the product along the detaching direction, in this case along +X direction. The reason for this step is to analyze the boundary information for the simulation process: how much movement along the detachment direction should be analyzed before the successful detachment of the component can be confirmed.

The third step sets up the transformation details (displacement and direction) through a transformation matrix and the detail definition of the matrix can be found in the Solidworks online API tutorial (Dassault Systems, 2016). The direction is positive X in this case and the displacement is set to 1.5 mm (a small displacement).

The fourth step starts the simulation process. Basically, the component under study is dragged along the detachment direction (+X) for a small distance (1.5 mm) and then the whole assembly is checked for interferences. If the number of interferences (being registered in the variable "counter") doesn't equal to zero, the program will stop and return false, which means the component cannot be detached along +X direction. If no interferences are identified, the procedure will go back to the beginning of the step 4 and another dragging transformation will be applied to the component (1.5 mm displacement along +X direction). The whole procedure will continue until we reach the maximum size of the product along +X direction: the component is completely outside of the remaining product. The simulation will stop at this point.

If the program does not return false during the simulation process (step 4), it means the component can be detached from the product along +X direction. Thus, the program will release the resource object (swApp and AssemblyDoc) and return true to the user.

PROCEDURE 1: INTERFERENCE TEST ALONG +X DIRECTION

```
function PositiveXInterferenceTest(String AssemblyFileName, String componentName)
1: Initiation: swApp, AssemblyDoc<- componentName
2: Identify size of the product
   boundary = swAssy.GetBox(1);
   SizeX = Math.Abs(boundary[0] - boundary[3]);
3: Set up transformation matrix
   swXform = (MathTransform)swMathUtil.CreateTransform(vXfm);
4: for i=0->sizeX, do
        drag (componentName, swXform);
        int counter=GetInterferenceCount ();
        if counter !=0
             return false;
        end if
    end for
5: Release Resource Object
6: Return true;
7: end function
```

Figure 4.14: Details of the Interference Test along +X Direction

Unconstrained Subassembly Detection

The goal of the unconstrained subassembly detection task is to carry out parallel disassembly whenever possible, which will achieve more efficient disassembly process compared to the sequential disassembly. The concepts of **ContactLoop** and **ContactLoopCluster** are used to define the unconstrained subassembly in an EOL product: An unconstrained subassembly is either a **ContactLoop** or a **ContactLoopCluster** in a product which satisfies the following conditions:

Definition of Unstrained Subassembly:

For each **ContactLoop** or **ContactLoopCluster**, check whether there exists an external edge, which links to a **ConnectingComponent** that does not belong to the **ContactLoop** or the **ContactLoopCluster**. If such edge cannot be identified, the **ContactLoop** or **ContactLoopCluster** is an unconstrained subassembly.

Figure 4.15 explains the concept with an example. The product in the current stage contains two unconstrained subassemblies:

- S1: Part1, Part5 and Part6
- S2: Part2, Part8, Part3, Part 9, Part4

S1 is a **ContactLoop** and S2 is a **ContactLoopCluster.** For S1, two external edges (e1 and e2) exist and neither of them is connected to a **ConnectingComponent** (Part2 is an **OridnaryComponent**). Similar observation can be made for S2: two external edges (e3 and e2) exist and both of them connect to an **OrdinaryComponent** Part1. Thus, S1 and S2 are unconstrained subassembly.

On the other hand, Part 8, Part 2 and Part 3 form another **ContactLoop** S3, which is not an unconstrained subassembly: S3 has an external edge e4 which connects to a **ConnectingComponent** (Part9) that does not belong to the **ContactLoop** S3.

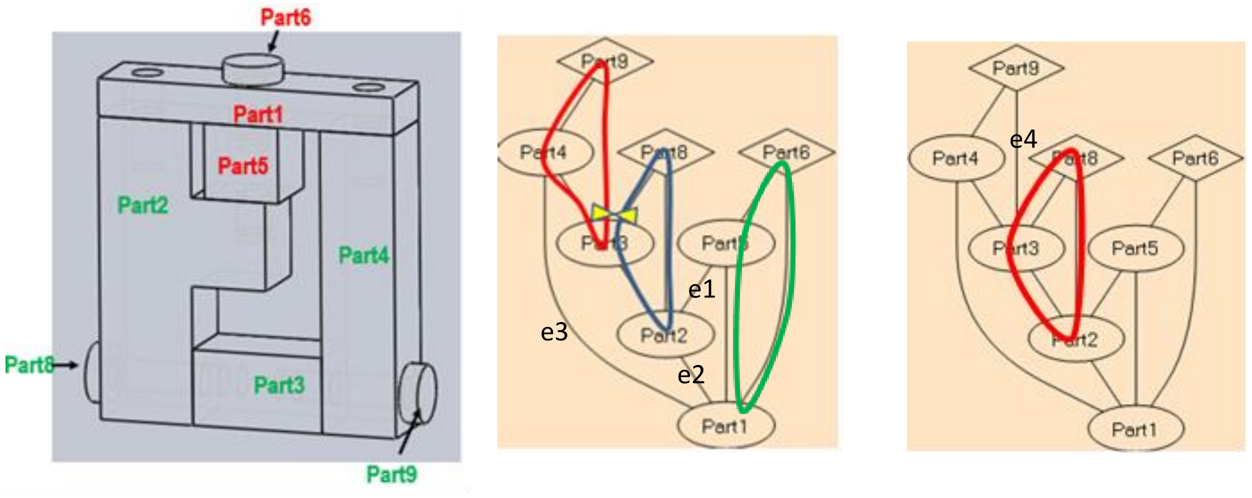


Figure 4.15: Unconstrained Subassembly Example

In order to identify the unconstrained subassembly in a product, information relating to the **ContactLoop** and the **ContactLoopCluster** should be provided. Different from other information in the DIM, which is static and can be determined a priori. The **ContactLoop** and **ContactLoopCluster** information is dependent on each of the disassembly states. Thus, it is not being identified in the beginning of the disassembly process, but rather being generated for each disassembly state dynamically. The detailed procedure can be broken into four functions as shown below.

F1->Identifying the Loops: The first function is to identify all the loops in the product connection graph, and the pseudo codes are presented in figure 4.16. The procedure starts with the function "findAllCycles", which takes each edge in the graph and pass it on to the sub function "findNewCycles" as input. The "findNewCycles" sub function then finds cycles that contains the input edge and returns the results.

PROCEDURE 2: IDENTIFYLING LOOPS IN A GRAPH

```
function findALLCycles(graph)
1. Initialize cycles = new static List<int[]>()
2. for i=0->graph. Length(0), do
        for j=0->graph. Length(1), do
            findNewCycles(new int[] {graph[i, j]})
        end for
    end for
end function


//Graph modelled as list of edges

//static int[,] graph = {{1, 2}, {1, 3}, {1, 4}, {2, 3}};
```

```
SubFunction findNewCycles(int[] path)
1. int n = path[0];
2. for i=0->graph. Length(0), do
3.      for y = 0->1, do
4.          if  graph[i , y] == n  // edge pointing to the current node
5.              x=graph [i, (y+1)%2]; //neighbor node
6.              if !visited(x, path); //neighbor not on the path yet
7.                  add to the path;
8.                  findNewCycles (path);
9.              end if
10.             if (path.Length >2) && (x==path[path.Length-1]) //Cycle found
11.                 int[] p = normalize(path);
12.                 int[] inv = invert(p);
13.                 if (isNew(p) && isNew(inv))
14.                     cycles.Add(p);
15.                 end if
16.             end if
17.         end if
18.     end for
19.  end for
20. End Sub function
```

Figure 4.16: Pseudo Code for Finding All the Loops in a Graph

In the sub function "findNewCycles", an outer loop scans all nodes of the graph and tries to locate neighborhood edge that connects to the input edge. The neighborhood edge will be further sent to the function "findNewCycles" to identify other connected edge (line 2 to line 8). The process will continue by recursively calling the sub function "findNewCycles", until a new cycle is found (the path is longer than two nodes and the next neighbor is the start of the path) (line 10). In order to avoid duplicate cycles, the identified cycles are normalized by rotating the smallest node to the start. Cycles in reversed ordering are also taken into account (Line 11 to Line 14).

F2->Identifying of the ContactLoop: The second function is to find all the **ContactLoops** from the loops identified from the previous procedure (F1). The main process is to apply each loop to the "isContactLoop" function and those return true from the "isContactLoop" function are the **ContactLoops**. Figure 4.17 shows the pseudo code for the "isContactLoop" function.

The "isContactLoop" function starts with checking whether only one **ConnectingComponent** exists in the cycle (Line 1 to line 8). If more than one **ConnectingComponent** or no **ConnectingComponent** is identified, the function will return false and the cycle is not a **ContactLoop.** From line 9 to line 26, the function tests whether all the **OrdinaryComponents** are connected to or has contact with the **ConnectingComponent** in the loop. It is done by retrieving all the edges of one **OrdinaryComponent** and check whether one or more of these edges further connect to the **ConnectingComponent** in the loop. If the checking returns true, the variable "counter" will increment by one. Such process will be applied to all the **OrdinaryComponents** in the loop and if in the end the value of the "counter" variable is less than the size of the input cycle by one, it means all the **OrdinaryComponents** are connected to the **ConnectingComponent** and the cycle is a **ContactLoop.**

PROCEDURE 3: CHECK WHETHER A CYCLE IS A CONTACTLOOP

```
function bool isContactLoop(List<String> cycle, Graph p1)
for i=0->cycle.Count(), do
   if (p1.FindNode(cycle[i]).Type==Type.connectingComponent
      numOfconnectingComponent++;
   end if
end for
if numOfconnectingComponent!=1
   return false;
end if
counter=0;
for i=0->cycle.Count(), do
   if (p1.FindNode(cycle[i]).Type == Type.ordinaryComponent
      List e1 = p1.FindNode(cycle[i]).OutEdges;
      List e2 = p1.FindNode(cycle[i]).InEdges;
      foreach edge in e1 and e2
          if edge.TargetNode.Id == connectingComponetInLoopID
             counter++;
          if e.SourceNode.Id == connectingComponetInLoopID
             counter++;
      end foreach
   end if
end for
if counter !=cycle.Count()-1
    return false;
end if
return true
end function
```

Figure 4.17: Pseudo Code for Determining Whether a Loop is a ContactLoop

F3->Identifying ContactLoopCluster: The third function is to find all the **ContactLoopClusters** and the main process steps are shown in figure 4.18. The procedure starts with initializing a "result" variable with empty initial value, for storing the identified **ContactLoopCluster.** Then, a searching over the **ContactLoops** is carried out (code section 2) to check whether two **ContactLoops** can be clustered to form a **ContactLoopCluster**. If no **ContactLoops** can be merged or only one merger happens, the procedure will end and the result will be returned (code section 3). If more than one mergers occur, the procedure further checks whether the resulting combinations can be further merged together to form a larger can cluster (code section 4). The process will continue until no new clusters can be found.

```
PROCEDURE 4: GENERATE CONTACTLOOPCLUSTER
function List<String> GenerateContactLoopCluster(List<String> contactLoops)
1.  Initialize empty List<String> result;
2.  for i=0->contactLoops.Count(), do
        for j=i+1 -> contactLoops.Count(),do
            if canCluster (contactLoop[i], contactLoop[j])
              add to result;
            end if
        end for
      end for
3.  if result.Count() == 0 or result.Count() == 1
        return result;
    end if
4.  if result.Count()!= contactLoop.Count()
        do
          tempt = result.ToList();
          size = result.Count();
          merge(result);
        while  (!sameList(tempt,result))
    end if
5.  return result
6.  end function
```

Figure 4.18: Pseudo Code for Identifying the ContactLoopCluster

<u>*F4->Identifying the Unstrained Subassembly:*</u> the last function is to identify all the unconstrained subassemblies of the product. The process checks all the **ContactLoops** and **ContactLoopClusters** identified in the previous steps and see if any of them loses constraints along certain disassembly directions. The function "IsUnConstrainedsubAssembly" is implemented for such purpose and figure 4.19 shows the major steps of the function.

The process is relatively simple and it recursively checks each node in the subassembly (the subassembly can be either a **ContactLoop** or a **ContactLoopCluster**). If none of the nodes is connected to an external **ConnectingComponent**, then it is an unstrained subassembly and the function returns true.

PROCEDURE 5: IS UNCONSTRAINED SUBASSEMBLY

```
function bool IsUnConstrainedsubAssembly(List<String> subAssy, Graph p1)
1.  for i=0->subAssy.Count(), do
       List e1=p1.FindNode(cycle[i]).Edges;
       foreach edge in e1
           if edge.TargetNode.Type == Type.ConnectingComponent
                        and (!subAssy.Contains(e.TargetNode.Id)
             return false;
           end if
       end foreach
     end for
5.  return true
6.  end function
```

Figure 4.19: Pseudo Code for the Function "IsUnConstrainedsubAssembly"

Overall Procedure for Finding All Feasible Disassembly Process Sequences

The overall procedure for finding all the feasible disassembly process sequences is shown in figure 4.20 in the next page. It utilizes the sub functions (F2->F4) discussed above and can be broken down into three parts. The first part is to pick any **ConnectingComponent** as a candidate component to be detached and apply the interference test to it. The procedure continues to the second part if the selected **ConnectingComponent** can pass the interference test. The second part is to carry out the component level stability and disassembility check, which searches if there exists an **OrdinaryComponent** which loses constraints due to the detachment of the **ConnectingComponent**. If so, we need to check whether this **OrdinaryComponent** can be detached without interferences. If there exists an interference, it means the detachment of the original **ConnectingComponent** will result in an unstable product state, in which some unconstrained component cannot be detached from the product. Such a situation is not allowed in the disassembly process and the program will thus reject the detachment of the candidate **ConnectingComponent** and start to test another candidate **ConnectingComponent.**

On the other hand, if the unconstrained **OrdinaryComponent** can be detached without interferences and the product can reach a stable state. The subassembly level stability and disassembility check will be further carried out, which checks if there exists an unconstrained subassembly that cannot be detached (cannot pass the interference test). If there is no unconstrained subassembly or the unconstrained subassembly can be detached without interference, the program will accept the disassembly plan and continue to the next iteration.

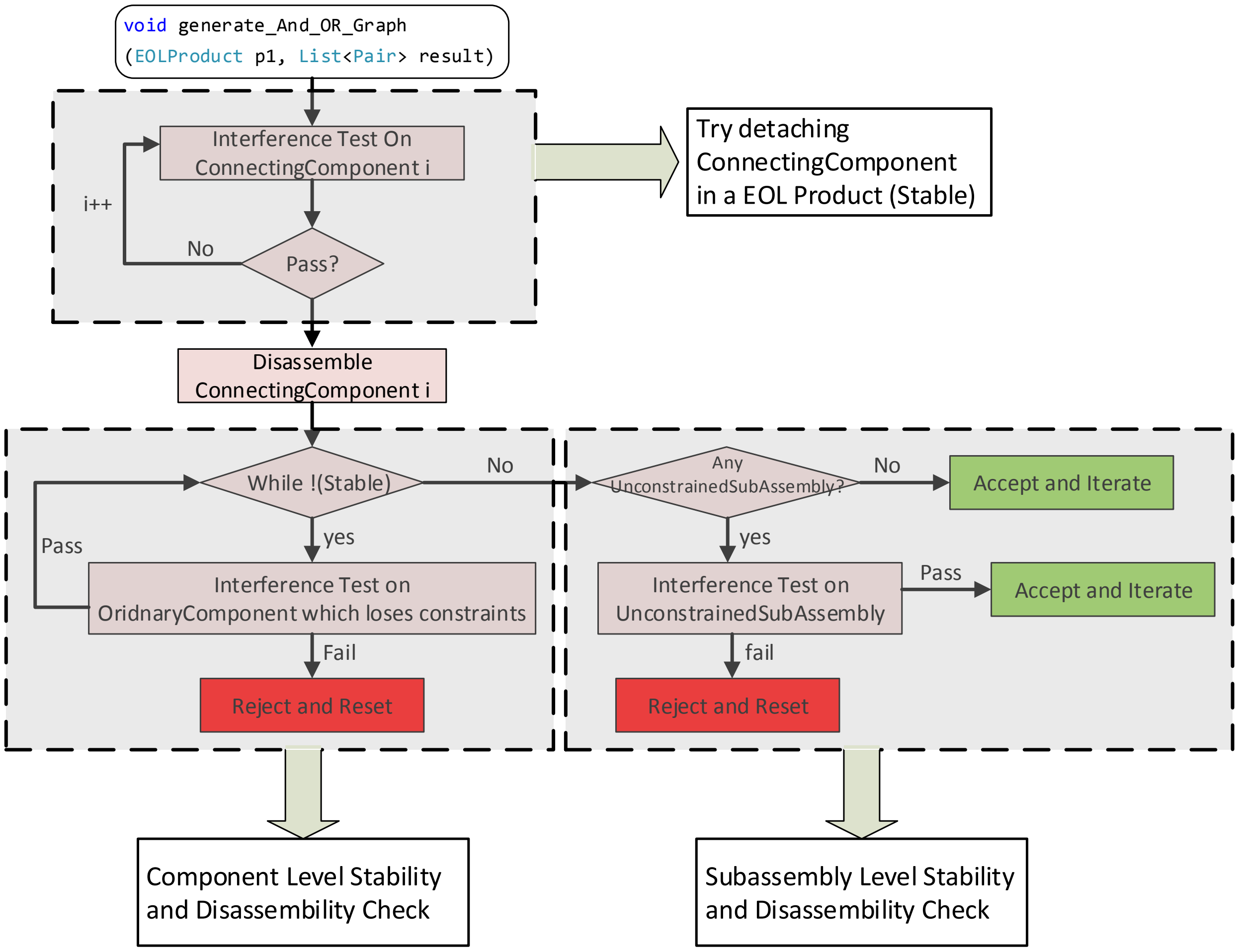

void generate_And_OR_Graph
(EOLProduct p1, List<Pair> result)
Interference Test On ConnectingComponent i
i++
No
Pass?
Try detaching ConnectingComponent in a EOL Product (Stable)
Disassemble ConnectingComponent i
While !(Stable)
No
Pass
yes
Interference Test on OridnaryComponent which loses constraints
Fail
Reject and Reset
Any UnconstrainedSubAssembly?
No
Accept and Iterate
yes
Interference Test on UnconstrainedSubAssembly
Pass
Accept and Iterate
fail
Reject and Reset
Component Level Stability and Disassembility Check
Subassembly Level Stability and Disassembility Check

Figure 4.20: Procedure for Finding All the Feasible Disassembly Process Sequences

4.4.2 LP based Disassembly Process Optimization

After identifying all the feasible disassembly process sequences, linear optimization can be applied to find the economically optimal sequence. The LP model gives the optimal solution based on maximizing the total value of the retrieved part/component and minimizing the total disassembly cost associated to them. Take figure 4.21 as an example, if we assign each disassembly operation (0, 1, 2, 3, 4, and 5) as a binary decision variable (x0, x1, x2, x3, x4, x5), the value we can retrieved from a set of disassembly operations is:

Value=V_{ABCDE} *(x0-x1-x2) +V_{ABCD} *(x1-x3) +V_{BCDE} *(x2) +V_{AB} *(x3-x4) + V_{CD} *(x3-x5) +V_A *(x2+x4) +V_B *(x4) +V_C *(x5) +V_D *(x5)

If we carry out only operations 0, 1, 3, and 4 (x0=x1=x3=x4=1, other equals to 0). The above equation tells us total value we can retrieved from such a disassembly plan is:

V_{ABCDE} *(1-1-0) +V_{ABCD} *(1-1) +V_{BCDE} *(0) +V_{AB} *(1-1) + V_{CD} *(1-0) +V_A *(0+1) +V_B *(1) +V_C *(0) +V_D *(0)

= V_{CD} + V_A + V_B

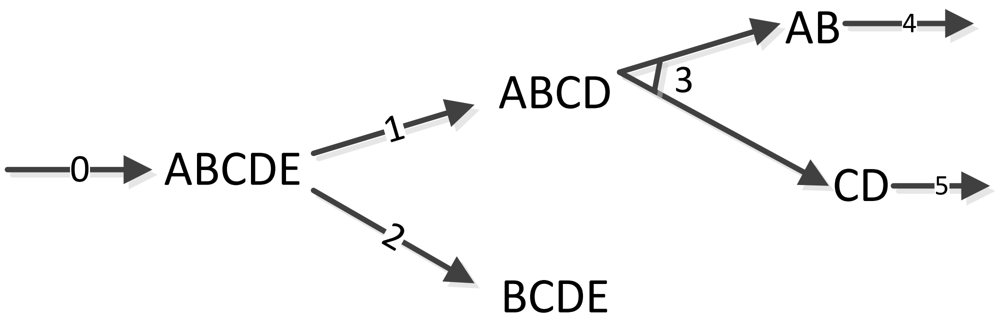


Figure 4.21: An Example of Four Parts

We can put into a generalized formulation as follows:

$$V = \sum_i \sum_j V_i * T_{i,j} * x_j$$

where T is the coefficient matrix and the value of the element in the matrix ($T_{i\,j}$) equals to -1, 0 or 1. The subscript j corresponds to operation and subscript i correspond to part or subassembly. If operation j disassembles subassembly i, T_{ij}= -1. If operation j assembles part i into a subassembly, T_{ij}=1. For other conditions, T_{ij}= 0. For the example in figure 4.21, the T matrix is as follows:

Table 4.2: Coefficient Matrix Example

	0	**1**	**2**	**3**	**4**	**5**
ABCDE	1	-1	-1	0	0	0
ABCD	0	1	0	-1	0	0
BCDE	0	0	1	0	0	0
AB	0	0	0	1	-1	0
CD	0	0	0	1	0	-1
A	0	0	1	0	1	0
B	0	0	0	0	1	0
C	0	0	0	0	0	1
D	0	0	0	0	0	1
E	0	0	0	0	0	0

Follow the same analysis for the disassembly operation cost, the complete disassembly LP model formulation is as follows:

Objective=V-C=$\sum_i \sum_j V_i * T_{i,j} * x_j - \sum_{j,k} C_{j,k} * y_{j,k}$

S.T.

$$\left.\begin{array}{l} 1.\ \sum_{in} x_{i,in} \geq \sum_{out} x_{i,out}\ , \forall\, i \\ \\ 2.\ \sum_{in} x_{i,in} \leq 1\ , \forall\, i \end{array}\right\} \boxed{\text{Constraints from AND/OR graph}}$$

$$\left.\begin{array}{l} 3.\ x_j = \begin{cases} 1, & if\ j = 0 \\ \sum_k y_{k,j}\ , & otherwise \end{cases} \\ \\ 4.\ \sum_k y_{k,j} = \sum_k y_{j,k}\ , \text{for } j = 1,2,3\ldots n \end{array}\right\} \boxed{\begin{array}{l}\text{Constraints from Task} \\ \text{Precedence graph}\end{array}}$$

where,
x_j *is a binary variable*, $\forall j$
$y_{j,k}$ *is a binary variable*, $\forall j, \forall k$

$x_{i,in}$ *is a binary variable*, $\forall i$
$x_{i,out}$ *is a binary variable*, $\forall i$

$i \in \{subassembly\ or\ component\ in\ the\ AND/OR\ \ graph\}$
$k, j \in \{0,1,2 \dots n\}$, where n represents the total number of operations

- Index i refers to a subassembly or a component in the AND/OR graph
- Index j refers to the disassembly operations, j=0,1,2…n, x_0 is a pseudo operation which represents the initialization of the disassembly process (EOL product is checked in). n represents the total number of operations.
- $x_{i,in}$ refers to one of the incoming flow variables (i.e. disassembly operation variable x_j) related to the subassembly (or the component) i.
- $x_{i,out}$ refers to one of the outgoing flow variables (i.e. disassembly operation variable x_j) related to the subassembly (or the component) i.
- $y_{k,j}$ refers to the disassembly operation j which is sequentially located after the disassembly operation k in the task precedence diagram.

Decision variables are x_j and $y_{j,k}$ and V_i and $C_{j,k}$ are constant coefficients representing the value of each part/component and the cost of disassembly operation. All of the decision variables are binary variables.

For every node in an AND/OR graph, the sum of the outgoing flow variables is equal to, or smaller than, the sum of the incoming flow variables (constraint 1). Also, only one path can be selected for a branch in the graph and thus the sum of the incoming flow variables should be less than 1 (constraint 2). As the initialization of the disassembly process, the decision variable of the first process x0 should be equal to 1 (Constraint 3). Constraint 3 also indicates that a disassembly operation variable related to operation j should be same on an AND/OR graph and a task precedence graph. Lastly, constraint 4 says for every node in the task precedence graph, the sum of the outgoing flow variables is equal to the sum of the incoming flow variables.

4.5 Case Study

This section verifies the Disassembly Sequence Generator application through an example (the graphical representation of the example is shown in figure 4.8). We start with the verification of the involved sub functions, which include: (1) loop detection, (2) **ContactLoop** detection, (3) **ContactLoopCluster** detection and (4) Unconstrained Subassembly detection, in section 4.5.1. The overall procedure for generating all the feasible disassembly sequences (shown in figure 4.20) is further validated in section 4.5.2. Lastly, in section 4.5.3, the LP-based optimization model is applied to the case study to find the optimal disassembly sequence.

4.5.1 Sub Function Verification

Figure 4.22 shows the implementation of functions for (1) loop detection, (2) **ContactLoop** detection, (3) **ContactLoopCluster** detection and (4) unconstrained subassembly detection. Applying those functions to the initial state of the product as shown in figure 4.8. The following results are obtained.

There are 41 cycles in the current connection graph, among which five are identified as **ContactLoop**:

L1: Part2->Part7->Part1 L2: Part2->Part8->Part3 L3: Part1->Part10->Part4

L4: Part1->Part6->Part5 L5: Part9->Part4->Part3

These **ContactLoops** can be further clustered to form one **ContactLoopCluster**:

"CLC1: Part2->Part7->Part1->Part8->Part3->Part10->Part4->Part6->Part5->Part9"

There is no unconstrained subassembly at this state.

If the disassembly operator detaches Part7 and Part10, the EOL product reach a new state. The results of the above sub functions are (shown in figure 4.23):

There are 17 cycles in the current connection graph, among which three are identified as **ContactLoop**:

L1’: Part2->Part8->Part3   L2’: Part1->Part6->Part5   L3’: Part9->Part4->Part3

These **ContactLoops** can be further clustered to form one **ContactLoopCluster**:

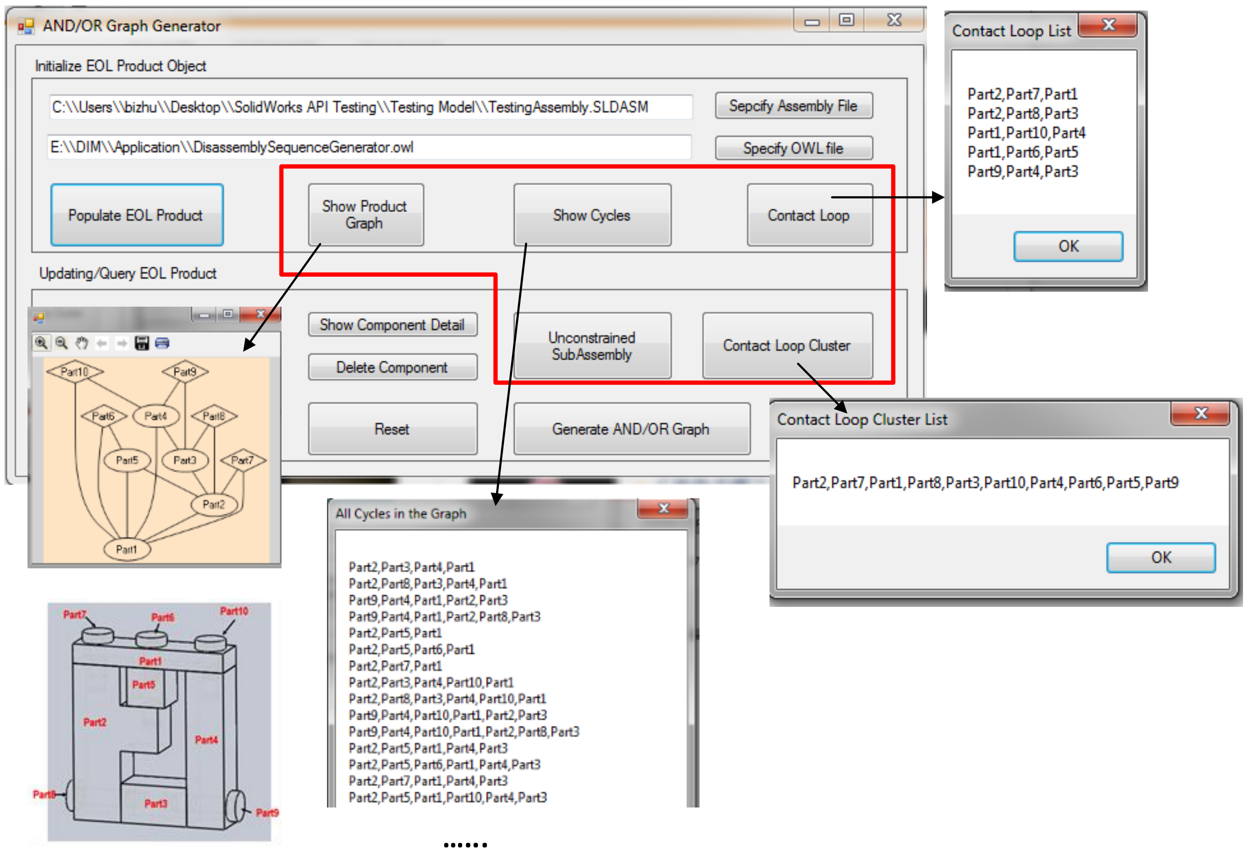
“CLC1’: Part2->Part8->Part3->Part9->Part4”

Also, there is two unconstrained subassemblies at this state and they are:

L2’: Part1->Part6->Part5

CLC1’: Part2->Part8->Part3->Part9->Part4

From the results above, it is evident that the implemented sub functions return results as expected and we thus can verify the proposed sub functions.



Contact Loop List

```
Part2,Part7,Part1
Part2,Part8,Part3
Part1,Part10,Part4
Part1,Part6,Part5
Part9,Part4,Part3
```

Contact Loop Cluster List

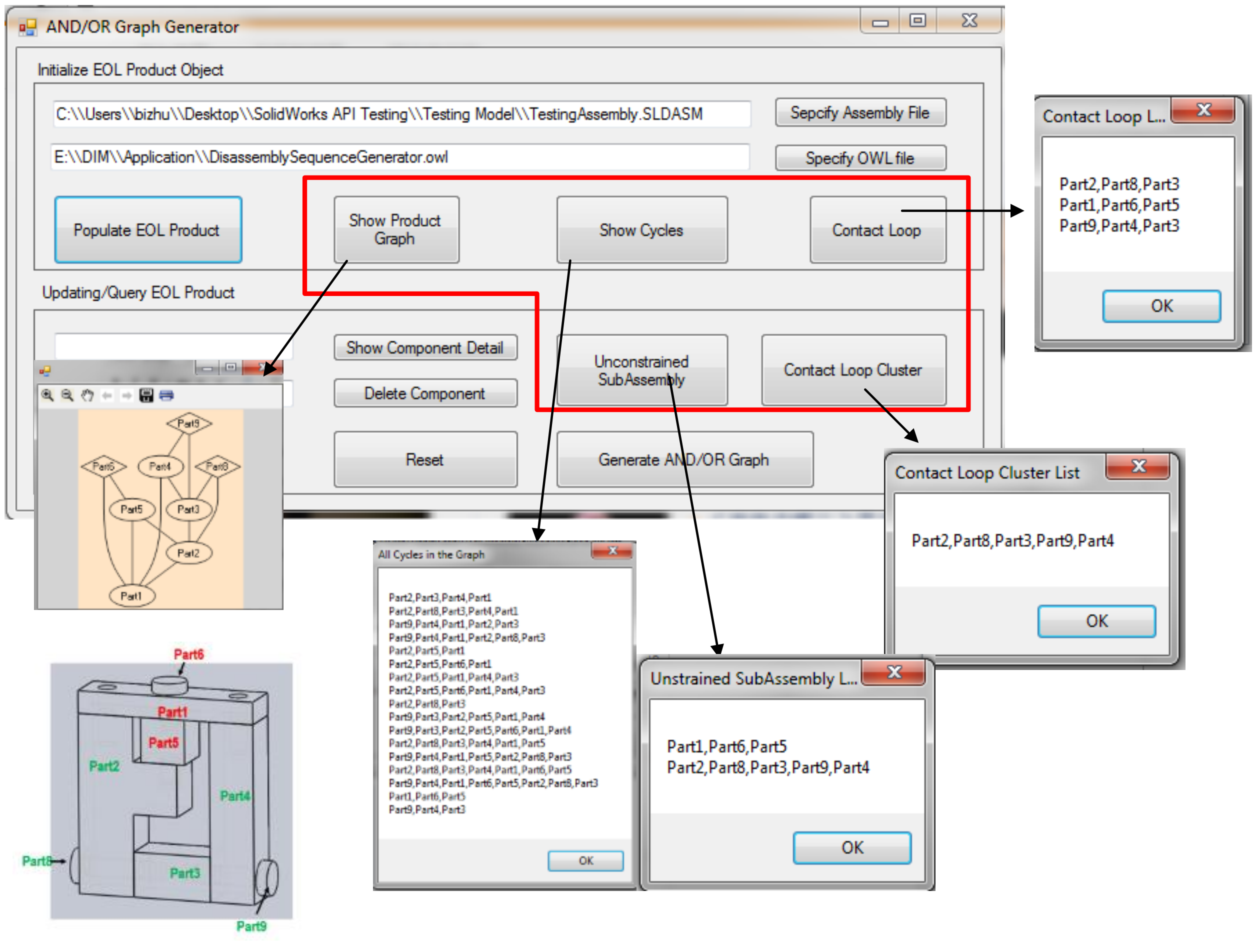
```
Part2,Part7,Part1,Part8,Part3,Part10,Part4,Part6,Part5,Part9
```

All Cycles in the Graph

```
Part2,Part3,Part4,Part1
Part2,Part8,Part3,Part4,Part1
Part9,Part4,Part1,Part2,Part3
Part9,Part4,Part1,Part2,Part8,Part3
Part2,Part5,Part1
Part2,Part5,Part6,Part1
Part2,Part7,Part1
Part2,Part3,Part4,Part10,Part1
Part2,Part8,Part3,Part4,Part10,Part1
Part9,Part4,Part10,Part1,Part2,Part3
Part9,Part4,Part10,Part1,Part2,Part8,Part3
Part2,Part5,Part1,Part4,Part3
Part2,Part5,Part6,Part1,Part4,Part3
Part2,Part7,Part1,Part4,Part3
Part2,Part5,Part1,Part10,Part4,Part3
```

……

Figure 4.22: Verification of Sub Functions at State 1 (Initial State)



```
C:\\Users\\bizhu\\Desktop\\SolidWorks API Testing\\Testing Model\\TestingAssembly.SLDASM
E:\\DIM\\Application\\DisassemblySequenceGenerator.owl
```

Contact Loop L...

```
Part2,Part8,Part3
Part1,Part6,Part5
Part9,Part4,Part3
```

Contact Loop Cluster List

```
Part2,Part8,Part3,Part9,Part4
```

All Cycles in the Graph

```
Part2,Part3,Part4,Part1
Part2,Part8,Part3,Part4,Part1
Part9,Part4,Part1,Part2,Part3
Part9,Part4,Part1,Part2,Part8,Part3
Part2,Part5,Part1
Part2,Part5,Part6,Part1
Part2,Part5,Part1,Part4,Part3
Part2,Part5,Part6,Part1,Part4,Part3
Part2,Part8,Part3
Part9,Part3,Part2,Part5,Part1,Part4
Part9,Part3,Part2,Part5,Part6,Part1,Part4
Part2,Part8,Part3,Part4,Part1,Part5
Part9,Part4,Part1,Part5,Part2,Part8,Part3
Part2,Part8,Part3,Part4,Part1,Part6,Part5
Part9,Part4,Part1,Part6,Part5,Part2,Part8,Part3
Part1,Part6,Part5
Part9,Part4,Part3
```

Unstrained SubAssembly L...

```
Part1,Part6,Part5
Part2,Part8,Part3,Part9,Part4
```

Figure 4.23: Verification of Sub Functions at State 2 (When Part7 and Part10 have been Detached)

4.5.2 Overall Procedure Verification

The overall procedure to generate all the feasible disassembly sequences has been shown in figure 4.20. Here, we apply the case study problem to the application to demonstrate the search process. Figure 4.24 below shows the details of one search iteration, which generates one feasible disassembly sequence.

The application procedure starts with picking any of the **ConnectingComponent** as the candidate to be detached. In this example, Part9 is selected and the interference test is applied on it to check whether Part9 can be detached without collisions with the other components. The result from the interference test will be true, which indicates no collisions will happen during the disassembly of Part9. Next step is to check the component and subassembly level stability/disassembility. Because the product reaches a stable state (state 2) after the detachment of Part9 (there exists no unstable components or subassemblies), the detachment of Part9 is accepted as feasible disassembly step.

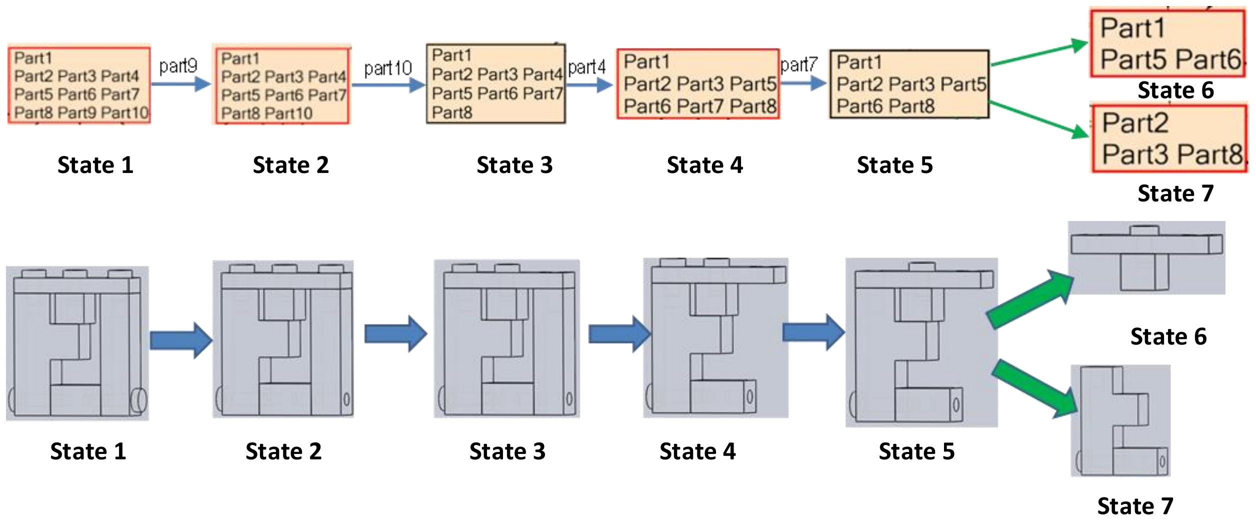

Part1 Part2 Part3 Part4 Part5 Part6 Part7 Part8 Part9 Part10
part9
Part1 Part2 Part3 Part4 Part5 Part6 Part7 Part8 Part10
part10
Part1 Part2 Part3 Part4 Part5 Part6 Part7 Part8
part4
Part1 Part2 Part3 Part5 Part6 Part7 Part8
part7
Part1 Part2 Part3 Part5 Part6 Part8
Part1 Part5 Part6
State 6
Part2 Part3 Part8
State 7
State 1
State 2
State 3
State 4
State 5
State 6
State 1
State 2
State 3
State 4
State 5
State 7

Figure 4.24: Process Description for Generating One Feasible Disassembly Sequence

At state 2, similar process will be applied. First, the application procedure will pick any of the **ConnectingComponent** as the candidate to be detached and Part10 is selected. Interference test will be further applied on Part10. However, in this state, even though Part10 can pass the interference test, it does not immediately accept the detachment of Part10 as a feasible disassembly process. It is because that the EOL product reached an unstable state (state 3) after Part10 is disassembled: the component level stability check will identify that component Part4 loses constraints along +x, -y, +z and –z directions and becomes unstable. Thus, further interference test on the unstable component Part4 should be carried out. In this case, Part4 can be detached and the EOL product will reach a stable state (State 4). Until this point, the application validates the feasibility of the disassembly process, "detachment of Part10" and disassembly process, “detachment of Part4”, and suggests they should be carried out sequentially in order to reach a stable state (State 4).

At state 4, similar process will be again applied and Part7 is selected as the candidate to be detached. The interference test on Part7 will be passed successfully and the EOL product will reach state 5 if Part7 is detached. However, the EOL product in state 5 contains subassembly level instability: two unstrained subassemblies (S1: “Part1, Part5, Part6” and S2: “Part2, Part3, Part8”) are identified. Thus, the interference test will be applied to them and parallel disassembly will be applied to yield two stable subassemblies (State 6 and State 7).

The searching process will iterate as above until all the feasible disassembly sequences are identified. The final result is shown in figure 4.25.

The result as shown in figure 4.25 identifies all the feasible disassembly sequences. We can see, with a 10 parts product, we can have theoretically 10! (3,628,800) disassembly sequences. However, the feasible disassembly sequences are much less (565 total for the case study problem) due to the geometrical and topological constraints.

A further study on those generated feasible disassembly sequences shows that they are 100% feasible in a real scenario, which means the application gives no error disassembly sequence that is geometrically or topologically impractical.

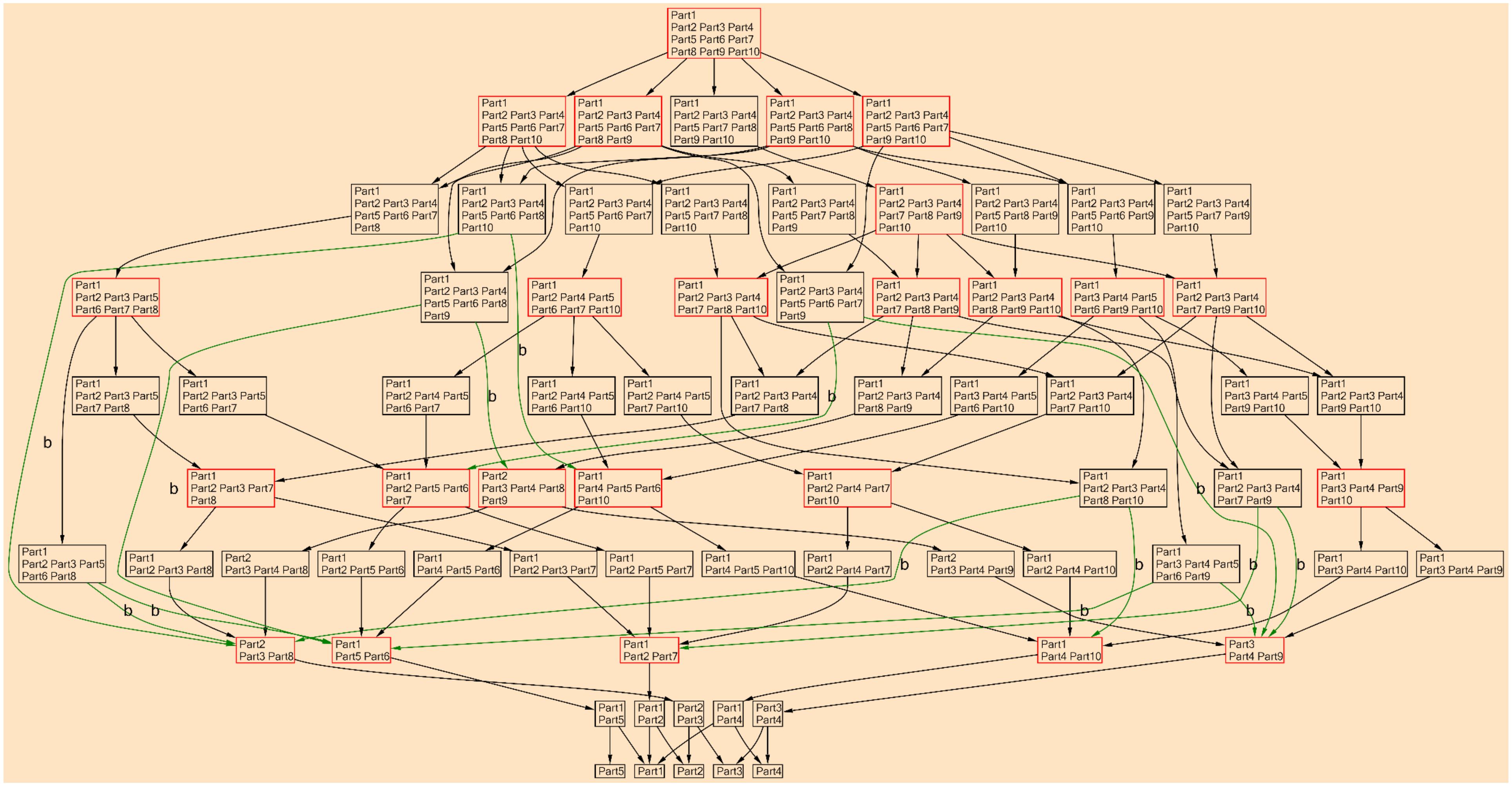


Figure 4.25: All the Generated Feasible Disassembly Sequences Related to the Case Study

4.5.3 LP-Based Optimization Model Verification

The LP-based optimization model can be applied to generate an economically optimal disassembly sequence. Instead of applying the LP model directly on the AND/OR graph as shown in figure 4.25, we pick a simpler example as shown in figure 4.26 below to validate the LP optimization model, with the intension to be more concise and clear.

Figure 4.26 represents a product with six parts (ABCDEF). All the feasible disassembly sequences are generated. Both total and partial disassembly is allowed as long as the profit is maximized. The cost of disassembly operation is known, and has given in a matrix form (figure 4.26); because, the cost of a certain disassembly operation is dependent on the previous operation. One single disassembly operation may cost differently depending on a particular disassembly sequence that has been previously followed till this operation. It means $C_{1,2}$ (cost of operation 2 carried out after operation 1) is different from $C_{3,2}$ (cost of operation 2 carried out after operation 3). Also the revenues of all the part, subassembly and assembly are known (figure 4.27). They can be positive numbers which means they have some values for reuse or recycling; they can also be negative numbers which means they can't be reused or recycled and maybe hazardous to the environment: so they have negative values. Given this information, an optimal disassembly sequence needs to be determined so that the profit will be maximized.

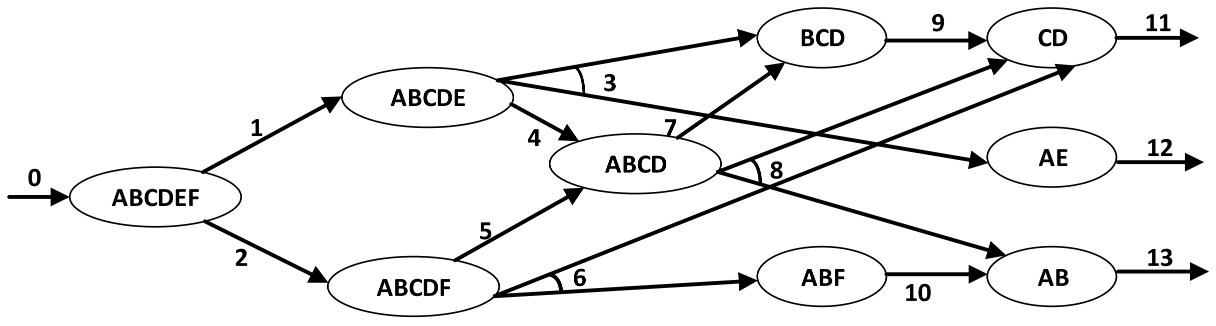


Figure 4.26: One Simple Generated Feasible Disassembly Sequences Example

Node	Value ($)
ABCDEF	-50
ABCDE	-35
ABCDF	-30
ABCD	200
ABF	220
BCD	-30
AB	170
AE	160
CD	250
A	152
B	78
C	180
D	220
E	160
F	130

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	70	69	68	67	66	65	64	63	62	61	60	59	58	57
2	50	51	52	53	54	55	56	57	58	30	20	10	20	15
3	60	50	51	36	15	20	21	22	23	24	25	26	27	28
4	30	49	52	26	16	16	18	20	22	24	26	28	32	40
5	50	48	53	27	17	17	36	15	30	10	5	20	30	41
6	45	47	54	28	18	18	25	40	45	50	60	30	32	42
7	30	46	55	29	19	19	11	30	25	16	11	18	15	43
8	29	30	56	30	20	20	30	31	32	33	34	8	30	44
9	28	29	57	34	21	21	31	32	33	34	40	42	31	45
10	27	28	30	45	22	22	32	15	20	20	10	30	32	46
11	26	27	28	46	23	30	33	15	10	20	16	14	33	47
12	25	26	36	47	30	35	34	30	25	51	50	9	34	40
13	24	25	23	48	35	26	35	60	14	70	60	45	35	30

Figure 4.27: The Value Vector and Cost Matrix for the Case Study Product

Applying LP optimization model proposed in section 4.4.2, the optimal path of the problem in figure 4.26 is shown in figure 4.28 below:

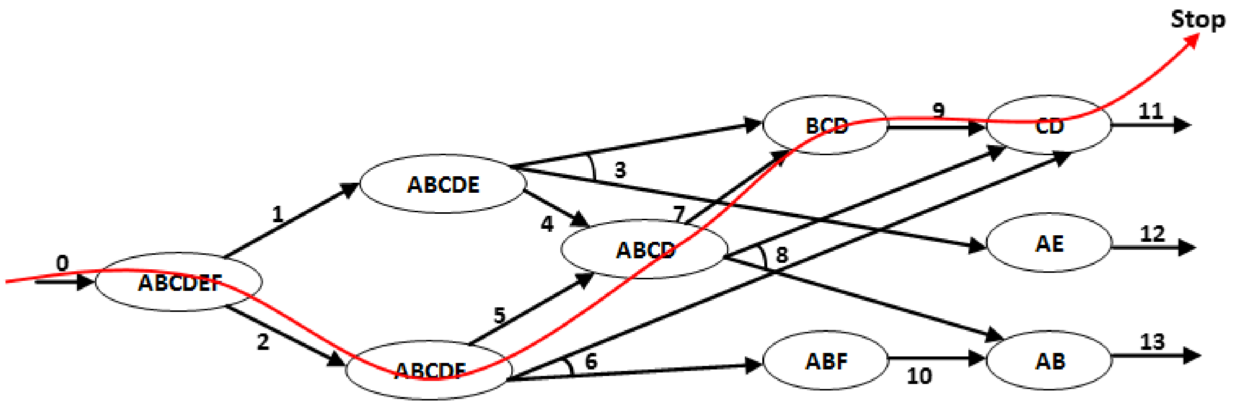


Figure 4.28: The Optimal Disassembly Path

In order to verify the model, the following two extreme cases are checked:

(1) Let us change the value of subassembly ABCD to extreme high. The program successfully stops at node ABCD for part reuse, as shown in figure 4.29. It means that subassembly ABCD is valuable enough for reuse and it should not be further disassembled.

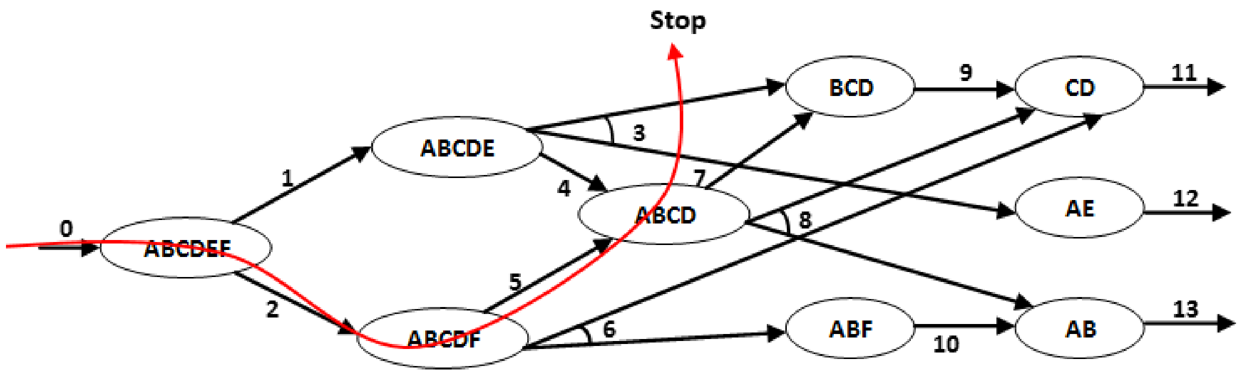


Figure 4.29: An Optimal Disassembly Path (Verification Scenario 1)

(2) Let us change the cost of disassembly operation C_{47} and C_{57} to very high values (big M). Because of the high operation costs associated to the operation 7, the optimal disassembly sequence will not continue through the arc 7 (which represents operation 7) and will detour to operation 8 instead, as shown in figure 4.30.

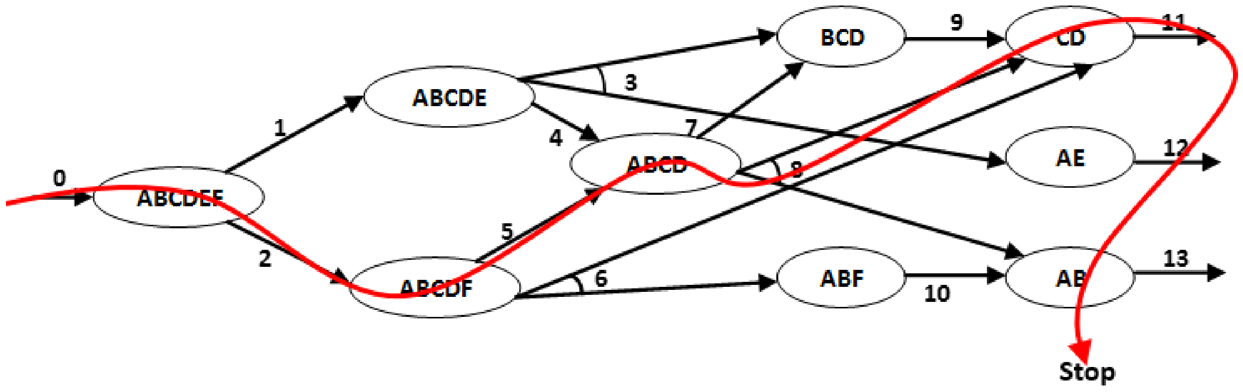


Figure 4.30: An Optimal Disassembly Path (Verification Case 2)

From the analysis above, we can conclude that the optimization model is quite convincing and it generates optimal disassembly sequence as expected.

Chapter 5 ADAPTIVE DISASSEMBLY PLANNING

Chapter 5 focuses on the problem of the adaptive disassembly planning, which considers the product and process uncertainties. We start with problem description in section 5.1. Next in section 5.2, the Adaptive Disassembly Planning Information Model is elaborated. *Detailed application method for carrying out the dynamic disassembly sequence generation is presented in section 5.3. Lastly, the chapter is wrapped up with a case study to verify the overall application procedure.*

5.1 Problem Definition

Adaptive disassembly planning considers all the feasible disassembly sequences as input and determines the optimal disassembly sequence. It takes the following two extra issues into consideration:

(1) Uncertainty issue: As mentioned in the previous chapters, unlike the assembly process, the disassembly process has various uncertainty issues. Thus, extra information and special mechanisms are needed for the uncertainty handling. Two types of uncertainties are considered in this dissertation: (1) Component/assembly function uncertainty and (2) Operation uncertainty.

Component/assembly functional uncertainty: each component or assembly might be associated with a primary function, which contributes to the product overall function. When an EOL product is at the end of its service life, its component or subassembly might not be functional, and such information is critical in the disassembly planning process. However, it could only be realized gradually during the disassembly process.

Operational uncertainty: during the disassembly process, certain operation, such as unscrewing, might not succeed due to the component's current physical conditions (the component may have deformed or corroded during its service period). Then, extra special operations are necessary to handle such situations, which will incur a higher cost. Since this information is also unknown at the beginning of the disassembly process, it is called the operation uncertainty.

(2) Degradation issue: Component/assembly degradation is also a critical factor in disassembly planning. Degradation is a gradual change in properties (like tensile strength, color, shape, etc.) of the component, which usually does not affect the overall function of a component until it reach to a critical point. However, degradation does affect the economic quantification of EOL product or component. For example, some subassembly might be functional, but the reuse value of the subassembly still could be lower than the expected average reuse value (the subassembly is close to failure) or higher than the expected average reuse value (the subassembly still has a long remaining useful life time).

In order to handle the above two issues, extra information is needed and it has been identified in chapter 3 (the Uncertainty Information Model and the Degradation Information Model in the domain layer). However, some of the involved information for a specific EOL product can hardly be acquired a priori (e.g. the condition of an internal component usually cannot be identified at the beginning of the disassembly process). Rather, this information is revealed gradually during a disassembly process. Thus, an “optimal” path is determined at each stage of the disassembly process with the limited information available at the current time and will be re-evaluated after reaching a new stage with more information identified. Thus, it is called adaptive disassembly planning problem.

5.2 Adaptive Disassembly Planning Information Model

The Adaptive Disassembly Planning Information Model is presented in this section. We start with the requirement analysis in section 5.2.1 and the formal Adaptive Disassembly Planning

Information Model is described in section 5.2.2 using the UML class diagram as a graphical notation.

5.2.1 Requirement Analysis for the Adaptive Disassembly Planning Information Model

The required information for handling the uncertainty and degradation issue has been identified in the domain level sub models (the Uncertainty Model and the Degradation Model). The Uncertainty Model is based on the Bayesian Network theory, whereas the Degradation Model is based on the Fuzzy Logic theory (refer to chapter 3 for details). From a high level view, the main information in both of the models is basically statistical information, which provides certain degrees of belief in the relevant issues. However, in order to make a disassembly decision, disassembly benefits (utility) and disassembly constraints should also be considered and a certain disassembly decision theory should be formed. In this dissertation, disassembly decision theory is defined as:

Disassembly Decision Theory = Probability Theory + Utility Theory + Disassembly Constraints

The fundamental idea of the disassembly decision theory is that a computer aided disassembly planner is rational if and only if it chooses the feasible disassembly action (satisfying all the constraints) that yields the highest expected disassembly utility, averaged over all the possible outcomes of the action. This is also called the principle of Maximum Expected Utility (MEU) in the traditional decision theory.

The realization of the Disassembly Decision Theory yields what we called Disassembly Decision Network (DDN) and it can be described formally as a six-tuple: *DDN= (P-DN, UTN, UN, TR, CPT, F), where*

Process Decision Node (P-DN): $P\text{-}DN = \{P\text{-}DN_1, P\text{-}DN_2, P\text{-}DN_3 \ldots, P\text{-}DN_N\}, N>0$, is a finite set of process decision nodes denoted by a rectangle shape. Each of the nodes can take two possible values ("carry out" or "don't carry out"), which represents the two choices available to the disassembly process planner regarding to a specific process decision.

Utility Node (UTN): *UTN= {UTN_1, UTN_2, UTN_3..., UTN_N}, N>0,* is a finite set of utility nodes denoted by a diamond shape and they are used to enable the numerical evaluation of the consequences of a decision. Two types of the utility nodes are further specified:

- _Process Utility Node (P-UTN)_: represents the cost that is associated with a disassembly process.
- _Disassembly Object Utility Node (D-UTN)_: represents the utility that is associated with a disassembly object, like component, subassembly, etc. The utility can be interpreted as the reuse value, recycling value or discard cost depending on the disassembly context (type of the disassembly object, whether or not the component is functioning, whether or not the subassembly is further detached, etc.)

Uncertainty Node (UN): *UN= {UN_1, UN_2, UN_3..., UN_N}, N>0,* is a finite set of uncertainty or chance nodes denoted by ellipse shapes and they are used to represent the random variables related to the problem. Two types of the uncertainty nodes are further specified:

- _Process Uncertainty Node (P-UN):_ is a variable representing whether or not a disassembly process is successfully carried out and two values are possible for this type of uncertainty node: {"success", "fail"}.
- _Disassembly Object Function Uncertainty Node (D-UN)_: is a variable representing whether or not a disassembly object is performing its designed function properly and two values are possible for this type of uncertainty node: {"function", "not function"}.

Transition Arc (TR): *TR= {TR_1, TR_2, TR_3 …, TR_N}, $N>0$*, is a finite set of directed arcs connecting different types of nodes. The intuitive meaning of a transition arc from node X to node Y is that X has a direct influence on Y, or there exists a causal relationship between X (cause) and Y (effect). Based on the types of the nodes to be connected, five types of TRs are further specified as follows:

- Type 1 *(P-DN → P-UTN)*: This type of transition arc connects a P-DN to a P-UTN, which describes the influences of a process decision on the process utility. In general, if the decision of a certain disassembly process is "carry out", then the utility (cost) of the relevant process is set to some negative value. On the other hand, if the decision of a certain disassembly process is "do not carry out", the relevant process utility (cost) should be zero.
- Type 2 *(P-UN → P-UTN)*: This is a transition arc connecting from a P-UN to a P-UTN and it describes the effect of the process uncertainty on the process utility. In general, if the disassembly process is successfully executed without problem (the value of P-UN is "success"), the process utility (cost) will be set to the average process cost. On the other hand, if the disassembly process fails, a higher process utility (cost) should be applied.
- Type 3 *(D-UN → D-UTN):* This is a transition arc connecting from a D-UN to a D-UTN and it describes the effect of the disassembly object function uncertainty on the disassembly object utility (refer to chapter 3 for the definition of disassembly object). In general, if the disassembly object is "not functioning", it means this disassembly object cannot be reused and thus the relevant utility is set to either the recycle value (if it is a component) or discard cost (if it is a subassembly). On the other hand, if the disassembly object is "functioning", it means this disassembly object can be reused and the relevant utility should be set to the reuse value.

- Type 4 *(P-DN → D-UTN)*: This is a transition arc connecting from a P-DN to a D-UTN and it describes the influences of a process decision on the disassembly object utility. In general, if a process disassembles the disassembly object, then the relevant utility is set to zero (the disassembly object doesn't exist anymore). Otherwise, the relevant utility will be set to either the reuse value, the recycling value or the discard cost depending whether the disassembly object is functioning properly.
- Type 5 *(D-UN →D-UN):* A transition arc connecting from a D-UN to a D-UN, which describes the function dependency between different disassembly objects. As an example, whether or not a computer is functioning properly is dependent on the functionality of its internal component, like CPU, motherboard, etc. It can be represented as: D-UN $_{\text{CPU}}$ → D-UN $_{\text{Computer}}$, D-UN $_{\text{motherboard}}$ → D-UN $_{\text{Computer}}$.

Conditional Probability Table (CPT): *$CPT = \{CPT_1, CPT_2, CPT_3..., CPT_N\}$, $N>0$*, is a finite set of conditional probability tables and each is attached to an uncertainty node described above. For each node, a CPT represents the conditional probability distribution $P(X_i|Parent(X_i))$, which quantifies the effect of the parents on the node. This is the statistical information, which has been included in the Uncertainty Model in the domain layer of DIM (refer to chapter 3 for detail).

Fuzzy model (F): *$F = \{F_1, F_2, F_3..., F_N\}$, $N>0$*, is a finite set of fuzzy models and each is attached to a Disassembly Object Utility Node (D-UTN) described above. It is used to quantify the degradation of disassembly objects by evaluating its real reuse value. The detail information related to the fuzzy model has been included in the Degradation Model in the domain layer of DIM (refer to chapter 3 for details.).

To summarize, the requirement for the Adaptive Disassembly Planning Information Model is to provide information elements to support the construction of the Disassembly Decision Network described above. Also, some information has been modeled in the domain layer sub models like Process Model, Uncertainty Model and Degradation Model. Thus, an integration of these models is also needed. Table 5.1 summarizes the modeling requirements for the Adaptive Disassembly Planning Information Model.

Table 5.1: Requirements for the Adaptive Disassembly Planning Information Model

R1	The modeling of different types of node: decision node, uncertainty node and utility node
R2	The modeling of five different types of transition arc.
R3	The linking to the uncertainty model and degradation model for the retrieval of relevant statistical information.

5.2.2 Formal Adaptive Disassembly Planning Information Model

The Adaptive Disassembly Planning Information Model deals with the information required for the adaptive disassembly planning problem. It is residing on the application layer of the DIM and is being extended based on three domain layer sub models named Process model, Uncertainty Model and Degradation Model. The overall structure is shown in figure 5.1 below. We will describe the model according to the information requirements identified above in the following sections.

R1->Node Modelling: Based on the definition of the Disassembly Decision Network, five classes representing different types of nodes have been modeled in the Adaptive Disassembly Planning Information Model: (1) class **Process_Decision_Node** representing P-DN, (2) class **Process_Utility_Node** representing P-UTN, (3) class **Process_Uncertainty_Node** representing P-UN, (4) class **Disassembly_Object_Function_Uncertainty_Node** representing D-UN, and (5) class **Disassembly_Object_Utility_Node** representing D-UTN.

R2->Transition Arc Modelling: Influences exist between different types of nodes and each of which form a certain transition arc in the Disassembly Decision Network. From the requirement analysis carried out in section 5.2.1, five types of transition arcs are identified and they are implemented in the Adaptive Disassembly Planning Information Model by introducing the object property "***influence***", which connects two nodes as its domain object and range object.

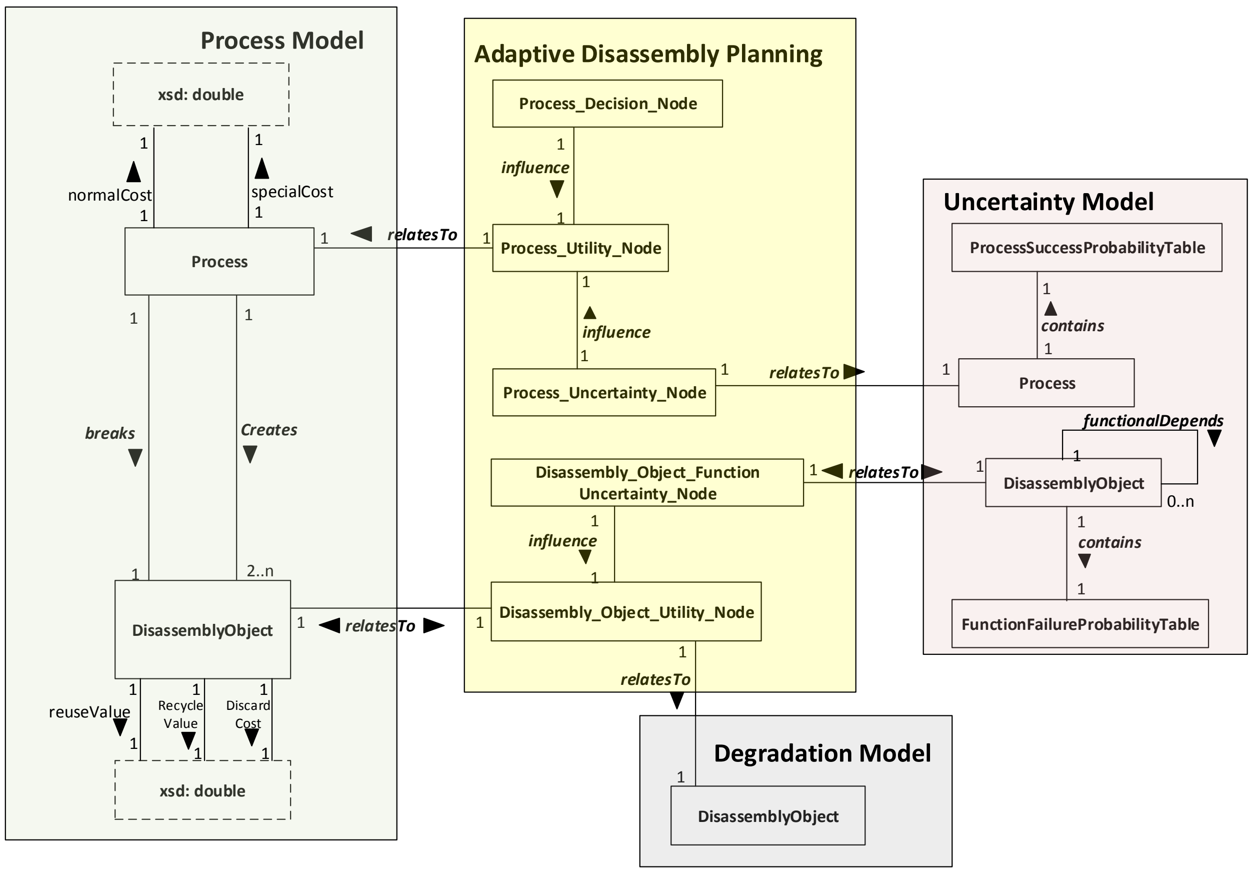


Figure 5.1: The Adaptive Disassembly Planning Information Model

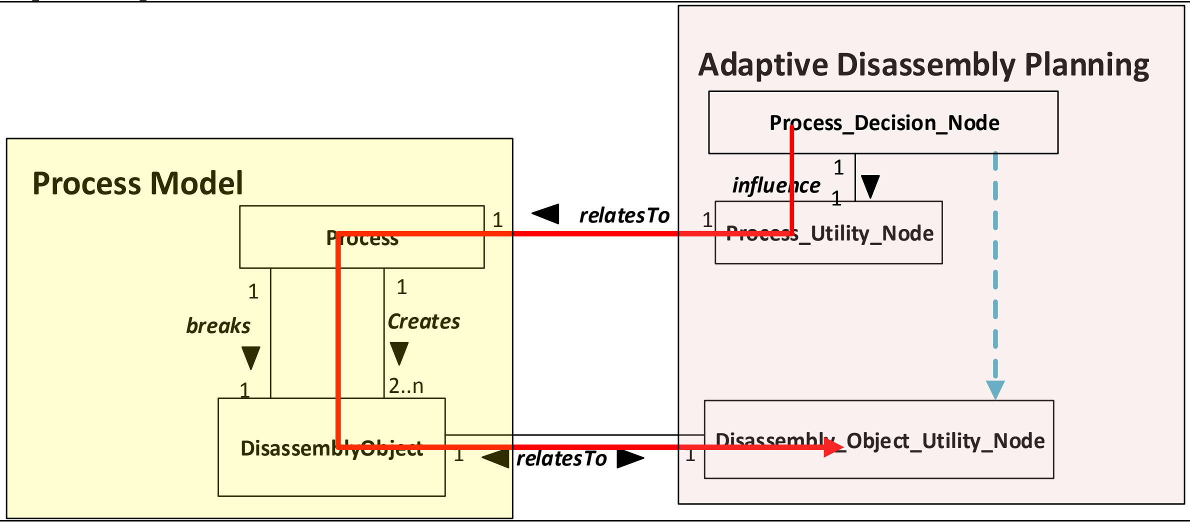
As shown in figure 5.1, three types of transition arcs (Type 1 to Type 3) have been explicitly defined. As an example, type 1 transition arc indicates the causal effect of a process decision (**Process_Decision_Node**) on the process utility (**Process_Utility_Node**). The two relevant nodes are related through the object property "***influence***": the domain of the "***influence***" object property is the **Process_Decision_Node** class, which indicates the cause, whereas the range of the "***influence***" object property is the **Process_Utility_Node** class, which indicates the effect of the cause.

A similar approach can be used to model Type 4 and Type 5 transition arc by introducing the "***influence***" object property to link from the **Process_Decision_Node** class to the **Disassembly_Object_Utility_Node** class (Type 4 *(P-DN → D-UTN)*); or to link from the **Disassembly_Object_Utility_Node** class to the **Disassembly_Object_Utility_Node** class (Type 5 *(D-UN →D-UN)*). However, such an approach will yield redundant or duplicate information due to the fact that the Type 4 and the Type 5 transition information have already been implicitly indicated in the domain level Process Model and Uncertainty Model. Thus, we utilize semantic rules to transfer such implicit information in the Process Model and Uncertainty Model to the explicit Type 4 and Type 5 causal information, which can be utilized to construct the Disassembly Decision Network. The detailed modeling mechanism is presented below:

<u>Modeling of Type 4 *(P-DN → D-UTN)* Transition Arc:</u> this type of transition arc describes the influences of a process decision on the utility of the relevant disassembly object. An in-depth study on this type of transition arc reveals that for a fixed process, the possible disassembly objects that can be influenced by it have already been modeled in the Process Model: A **Process** has influences on several **DisassemblyObjects** through the object property "***breaks***" and the object property "***creates***" (refer to the Process Model in chapter 3 for detailed description). Thus, the relationship between **Process** and **DisassemblyObject** in the Process Model actually indicates the influences of **Process_Decision_Node** on the **Disassembly_Object_Utility_Node.**

Thus, we don't need to explicitly include that relationship in the Adaptive Disassembly Planning Information Model; rather the following semantic rule (shown in table 5.2) has been added to transfer the relationship between the **Process** class and the **DisassemblyObject** class in the Process Model to the influence of the **Process_Decision_Node** on the **Disassembly_Object_Utility_Node** in the Adaptive Disassembly Planning Information Model:

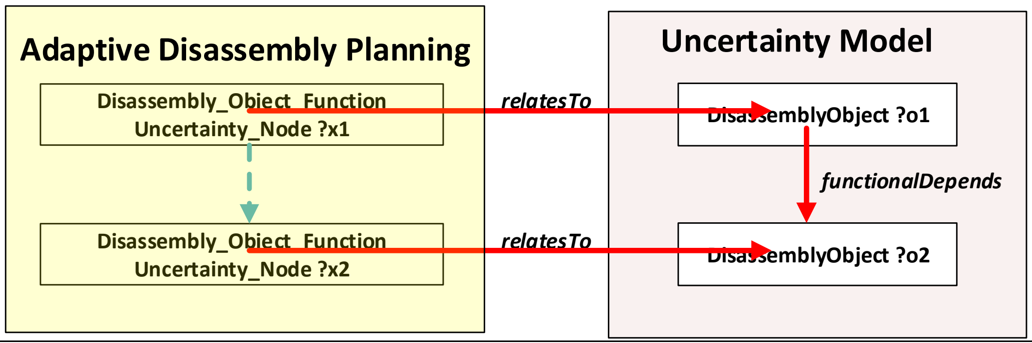
Table 5.2: Semantic Rule R1 Definition

Semantic Rule: R1	
Antecedent (red line)	**Consequent (blue line)**
Process_Decision_Node(?x), Process_Utility_Node(?y), Process(?z), DisassemblyObject(?d), Disassembly_Object_Utility_Node(?u), influence(?x, ?y), relatesTo(?y, ?z), (*breaks* (?z, ?d) or creates (?z, ?d)), relatesTo(?d, ?u)	influence (?x, ?u)
Graphical Explanation	
 Adaptive Disassembly Planning Process_Decision_Node influence 1 1 Process_Utility_Node Process Model Process 1 relatesTo 1 1 breaks 1 1 Creates 2..n DisassemblyObject 1 relatesTo 1 Disassembly_Object_Utility_Node	

Modeling of Type 5 *(D-UN →D-UN)* Transition Arc: this type of transition arc describes the functional dependency between different disassembly objects. Refer to figure 5.1, such information has already been modeled in the Uncertainty Model through "***functionalDepends***" object property. Thus, the following semantic rule (shown in table 5.3) has been included to transfer the relevant information in the Uncertainty Model to the Adaptive Disassembly Planner

Information Model, for representing the causal relationship between two **Disassembly_Object_Function_Uncertainty_Nodes.**

Table 5.3: Semantic Rule R1 Definition

Semantic Rule: R2	
Antecedent **(red line)**	**Consequent** **(blue line)**
Disassembly_Object_Function_Uncertainty_Node (?x1), Disassembly_Object_Function_Uncertainty_Node (?x2), DisassemblyObject(?o1), DisassemblyObject(?o2), relatesTo(?x1, ?o1), relatesTo(?x2,?o2), functionalDepends(?o1, ?o2)	influence (?x1, ?x2)
Graphical Explanation	
 Adaptive Disassembly Planning Disassembly_Object_Function Uncertainty_Node ?x1 Disassembly_Object_Function Uncertainty_Node ?x2 relatesTo relatesTo Uncertainty Model DisassemblyObject ?o1 functionalDepends DisassemblyObject ?o2	

R3->Model Integration: the implementation of this requirement has been partially shown in the previous discussion. The Adaptive Disassembly Planning Information Model links the Process model, Uncertainty model and Degradation model through object property "***relatesTo***". In detail, the integration is implemented in the following four places:

(1) The **Process_Uncertainty_Node** class in the Adaptive Disassembly Planning Information Model links to the **Process** class in the Uncertainty Model, through which the relevant process related Conditional Probability Table (**ProcessSuccessProbabilityTable**) information can be retrieved.

(2) The **Disassembly_Object_Function_Uncertainty_Node** class in the Adaptive Disassembly Planning Information Model links to the **DisassemblyObject** class in the Uncertainty Model, through which the relevant function related conditional probability table (**FunctionFailureProbabilityTable**) information can be retrieved.

(3) The **Process_Utility_Node** class in the Adaptive Disassembly Planning Information Model links to the **Process** class in the Process Model, through which regular and special process costs can be retrieved.

(4) The **Disassembly_Object_Utility_Node** class in the Adaptive Disassembly Planning Information Model links to the **DisassemblyObject** class in the Process Model, through which the related recycle value, reuse value and discard cost can be retrieved.

The Adaptive Disassembly Planning Information Model has been fully implemented in OWL and table 5.4 below summarizes the major model concepts.

Table 5.4: DIM OWL Implementation Concept Summarization

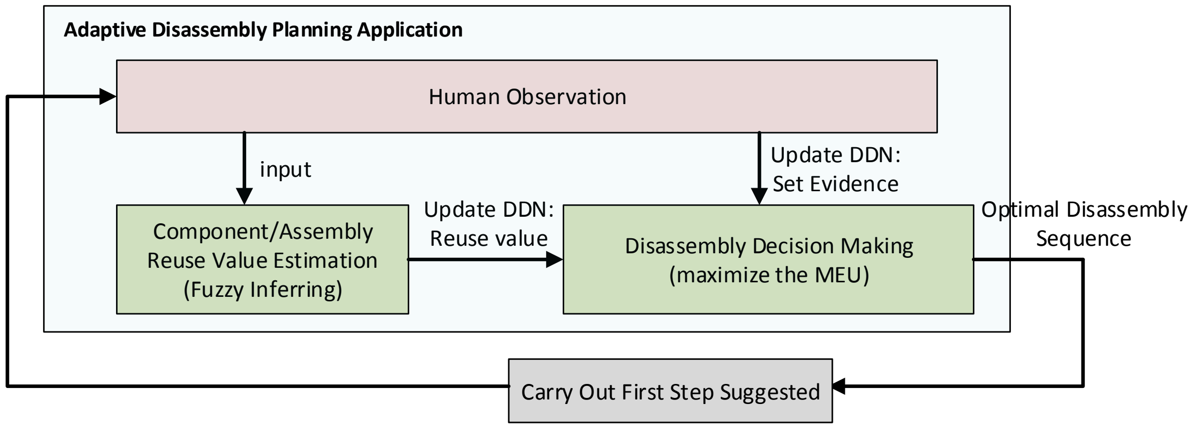
Model	**Class**	**Class Axioms**	**Object Property**
Adaptive Disassembly Planning.owl	Disassembly_Object_ Utility_Node	relatesTo **exactly** 1 (Process model: DisassemblyObject)	relatesTo
		relatesTo **exactly** 1 (Degradation model: DisassemblyObject)	
	Disassembly_Object_ Function_Uncertainty_Node	influence **exactly** 1 Disassembly_Object_Utility_Node	influence
		relatesTo **exactly** 1 (Uncertainty model: DisassemblyObject)	
	Process_Uncertainty_Node	influence exactly 1 Process_Utility_Node	
		relatesTo **exactly** 1 (Uncertainty model: Process)	
	Process_Utility_Node	relatesTo **exactly** 1 (Process model: Process)	
	Process_Decision_Node	influence **exactly** 1 Process_Utility_Node	
Imported Model			
Process.owl			
Degradation.owl			
Uncertainty.owl			

5.3 Adaptive Disassembly Planning Application

This section discusses the detailed procedure for solving the adaptive disassembly planning problem. A high level view of the procedure is shown in figure 5.2, which is an iterative process involving two major sub-functions (indicated as green boxes in figure 5.2):

F1->Component/Assembly Reuse Value Estimation. This function uses a fuzzy logic based approach for the component/assembly reuse value estimation. It takes the inputs from the human observation and further calculates the reuse value of the component or assembly and updates the Disassembly Decision Network accordingly.

F2->Disassembly Decision Making. This function carries out the Disassembly Decision Network based sequence optimization. It takes two types of information as inputs: (1) the component/assembly reuse value (the output of F1) and (2) the human observation on whether or not a certain component/subassembly is functioning properly. The output will be an optimal disassembly sequence, based on the current available information.



Adaptive Disassembly Planning Application
Human Observation
input
Update DDN: Set Evidence
Component/Assembly Reuse Value Estimation (Fuzzy Inferring)
Update DDN: Reuse value
Disassembly Decision Making (maximize the MEU)
Optimal Disassembly Sequence
Carry Out First Step Suggested

Figure 5.2: High Level View of the Adaptive Disassembly Planning Application

After an optimal disassembly sequence is generated, the disassembly operator will carry out the first step in the suggested optimal disassembly sequence, which will yield a new disassembly state.

New observation might be identified in the new disassembly state, which could affect both of the sub functions (F1 and F2). Thus, F1 and F2 will be re-evaluated based on the new observations and a new optimal disassembly sequence will be suggested. The whole process will iterate until the product is fully disassembled or the optimal disassembly plan becomes stable (remain same between iterations).

The following sections are organized as follows: two sub functions (F1 and F2) are discussed in detail in section 5.3.1 and section 5.3.2 first. Then, section 5.3.3 presents the complete application procedure in detail.

5.3.1 Component/Assembly Reuse Value Estimation

The goal of the first sub function is to estimate the component/assembly reuse value, which is an important piece of information for constructing the Disassembly Decision Network. A concrete mathematical model to quantify this information is challenging and is very much case dependent. Thus, we use the idea of fuzzy inference, a technique that facilitates the modeling of a complex system without the knowledge of its mathematical description, for the reuse value estimation. In general, the fuzzy inference system consists of four modules as indicated in the figure 5.3 below.

Fuzzification module: transforms the system inputs, which are crisp numbers, into fuzzy sets by applying the fuzzification functions. The system inputs are the critical variables identified in the Degradation Model and they are component/assembly age, market demand and a set of conditional parameters. It is assumed that these variables are sufficient to evaluate the component/assembly reuse value (refer to chapter 3 for detailed discussion on these input variables).

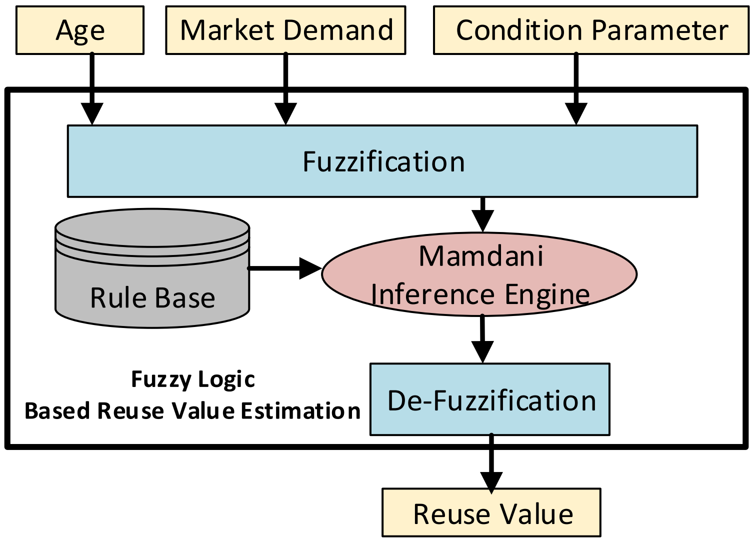


Figure 5.3: High Level View of the Fuzzy Inference System

Each of the input variable is modelled as a linguistic variable, which is a composite data structure containing a set of fuzzy terms. Each fuzzy term further contains: (1) a linguistic value (values are words in a natural or artificial language, e.g. Age = "Low") which an input variable can take and (2) a membership function, which is used to quantify the degree of truth (0 to 1) of classifying a certain numerical value (e.g. Age = 2.5 years) into the linguistic value (e.g. Age = "Low") the fuzzy term represents.

The following is the xml code for the "Age", "Market Demand" and "Operation Noise" (condition parameter) linguistic variable.

```
  <Variable VariableName="Age" LowerLimit="0" UpperLimit="5" VariableType="Input">
    <FuzzyTerm Name="Low" FunctionType="NormalMembershipFunction" Parameters="0,1.2"></FuzzyTerm>
    <FuzzyTerm Name="Medium" FunctionType="NormalMembershipFunction"
Parameters="2.5,1"></FuzzyTerm>
    <FuzzyTerm Name="High" FunctionType="NormalMembershipFunction"
Parameters="5,1.2"></FuzzyTerm>
  </Variable>
  <Variable VariableName="OperationNoise" LowerLimit="0" UpperLimit="50" VariableType="Input">
    <FuzzyTerm Name="Normal" FunctionType="NormalMembershipFunction"
Parameters="0,13"></FuzzyTerm>
    <FuzzyTerm Name="Abnormal" FunctionType="NormalMembershipFunction"
Parameters="50,13"></FuzzyTerm>
  </Variable>
  <Variable VariableName="MarketDemand" LowerLimit="0" UpperLimit="200" VariableType="Input">
    <FuzzyTerm Name="Low" FunctionType="NormalMembershipFunction" Parameters="0,50"></FuzzyTerm>
    <FuzzyTerm Name="High" FunctionType="NormalMembershipFunction"
Parameters="200,52"></FuzzyTerm>
  </Variable>
```

Knowledge base: stores IF-THEN rules provided by experts. In this dissertation, four general rules related to the high reuse value and low reuse value are modeled:

- Low Reuse Value Rule
  - R1: Age is High => Reuse Value will be Low
  - R2: Market Demand is Low => Reuse Value will be Low
  - R3: Condition Parameter is Worse => Reuse Value will be Low
- High Reuse Value Rule
  - R4: Age is Low and Market Demand is High and Condition Parameter is Normal => Reuse Value will be High

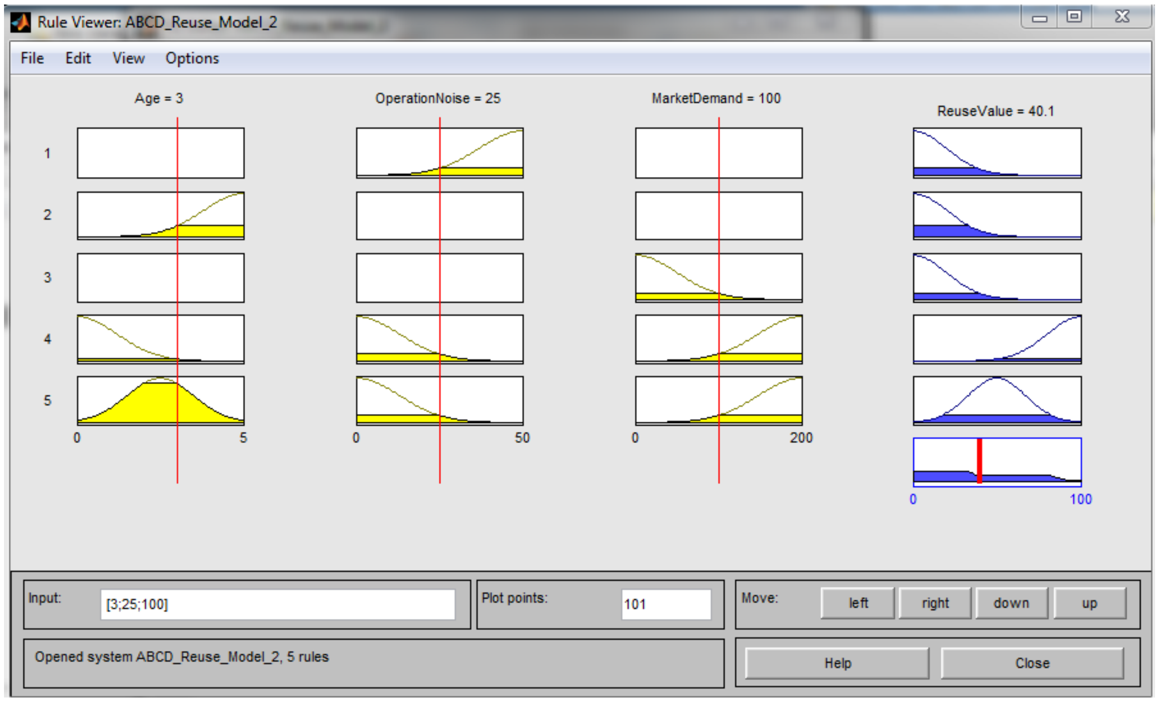
Other customized rules can be added if needed and they can be acquired from the Degradation Information Model. An example is shown below:

- Average Reuse Value Rule
  - Age is Medium and Market Demand is High and Condition Parameter is Normal => Reuse Value will be Average

Inference engine and Defuzzification: Fuzzy inference engine is the main decision making module in a fuzzy inference system. Its main operation is to convert the input fuzzy set into an output fuzzy set through an inference process. Whereas, the defuzzification process transforms the fuzzy set obtained by the inference engine into a crisp value.

In this dissertation, we use the popular Mamdani method for implementing the inference procedure and the details of the Mamdani method can be found at (Vukadinovic, 2013). The defuzzification process is based on the idea of "Centroid of Area", which returns the center of the area under the aggregated curve. If we think of the area as a plate of equal density, the centroid is the point along the x axis about which this shape would balance.

The sub function is implemented in Matlab and figure 5.4 shows the implementation of the method for the reuse value estimation of subassembly ABCD.



Input: [3;25;100]

Plot points: 101

Move:

Opened system ABCD_Reuse_Model_2, 5 rules

Figure 5.4: Fuzzy Influence Implementation in Matlab

As shown in figure 5.4, the crisp input is [age= 3, Operation Noise=25, Market Demand=100]. Five fuzzy control rules are defined (four general and one customized, not shown in the figure 5.4) and each will generate the fuzzy value of the output variable (ABCD's reuse value). The five generated fuzzy values will then be aggregated and further again be translated into crisp value, the final inferred result is 40.1 (reuse value of subassembly ABCD).

5.3.2 Disassembly Decision Network based Disassembly Planning

This section introduces a Disassembly Decision Network based adaptive disassembly planning approach, which integrates the Bayesian probability theory and the maximum expected utility (MEU) principle, for dynamically generating the optimal product disassembly sequence.

The determination of optimal disassembly sequence is to decide the value of each of the Process Decision Node (P-DN), which can yield a maximum disassembly object utility and a minimum the process utility (cost). If we annotate d_i as one possible disassembly sequence, then d_i can be expanded as follows:

$$d_i = pDN_i = \{pDN_{1,i}, pDN_{2,i}, pDN_{3,i} \dots pDN_{N,i}\}$$

Where pDN is short for P-DN. The first subscript represents the index of the process decision node in the Disassembly Decision Network, whereas the second subscript indicates the decision value ("carry out" or "do not carry out") associated to that node.

Then, the expected utility (EU) of a disassembly plan d_i is given by:

$$EU(d_i|e) = \sum_{p=1}^{n}\sum_{j=1}^{2}\{P(pUN_{p,j})|e\} * pUTN_p(d_i, pUN_{p,j}) + \sum_{o=1}^{k}\sum_{j=1}^{2}\{P(dUN_{o,j})|e\} * dUTN_o(d_i, dUTN_{o,j})$$

- $pUN_{p,j}$ represents the Process Uncertainty Node in DDN. The first subscript represents the index of the process uncertainty node in the Disassembly Decision Network, whereas the second subscript indicates the uncertainty value ("fail" or "success") associated to that node.
- $pUTN_p$ represents the Process Utility Node in DDN. The subscript represents the index of the process associated to that node. $pUTN_p$ can take different values depending on the arguments d_i and $pUN_{p,j}$.

- $dUN_{o,j}$ represents the Disassembly Object Function Uncertainty Node in DDN. The first subscript represents the index of the Disassembly Object Function Uncertainty Node in the Disassembly Decision Network, whereas the second subscript indicates the uncertainty value (“function” or “not function”) associated to that node.
- $dUTN_o$ represents the Disassembly Object Utility Node in DDN. The subscript represents the index of the disassembly object. $dUTN_o$ can take different values depending on the arguments d_i and $dUTN_{o,j}$.
- e is set of evidence identified during the disassembly process.

The above equation describes the expected utility (EU) of a disassembly option d_i given a set of evidences. It is an aggregation of two parts: (1) the expected utility of the disassembly process and (2) the expected utility of the disassembly object. Both of the parts are evaluated by calculating the summation of the relevant utilities, weighted over the probability values of the relevant uncertainty node.

In order to calculate the probability values like $P\left(dUN_{o,j}\right)|e$ and $P\left(pUN_{p,j}\right)|e$, Bayes rules are used here. In general, the basic task is to compute the posterior probability for a set of **query variables** (X) (in our case X is either $dUN_{o,j}$ or $pUN_{p,j}$), given some observed event—that is, some assignment of values to a set of **evidence variables** (e).

$$P(X|e) = \frac{P(X,e)}{P(e)} = \alpha P(X,e) = \alpha \sum_{y} P(X,e,y)$$

Y denotes the non-evidence, non-query variables $Y_1, Y_2 \ldots, Y_l$ (called the **hidden variables**) and α is the normalization constant.

Finally, the best decision D*, given the probability distribution and the utility model is given by:

$$D_* = max\ EU(d_i|e)$$

The above equation indicates that the optimal disassembly sequence d_{i*}, is a decision sequence which maximizes the $EU(d_i|e)$.

5.3.3 The Complete Adaptive Disassembly Planning Procedure

The disassembly sequence (disassembly plan) generated in section 5.3.3 can only be considered "optimal" at the current disassembly state, in which only limited information or evidence can be identified. Carrying out one disassembly operation according to the plan puts forward the disassembly object to a new state with possibly more evidences revealed, which can change the "optimal" disassembly result generated by the DDN based Disassembly Planning function. Thus, the complete adaptive disassembly planning procedure is developed here (figure 5.5) to iteratively generate optimal disassembly sequence.

When an EOL product is taken into the disassembly facility, function testing is applied first, which will provide certain evidences on whether certain component/assembly is working properly or not. Notice that the function testing at this stage can only provide the functionality information about some of the components or assemblies. Whether or not the other components or assemblies are functioning properly is still uncertain to the disassembly operator. However, the probability of them to be functional is updated based on the updated evidence. As an example, if the disassembly operator identifies that a fan assembly is functional and updates that information to the DDN as evidence, the probability of the motor to be functioning will be changed as follows:

$$P(Motor = functioning) \rightarrow\ P(Motor = functioning\ |Fan\ Assembly = functioning)$$

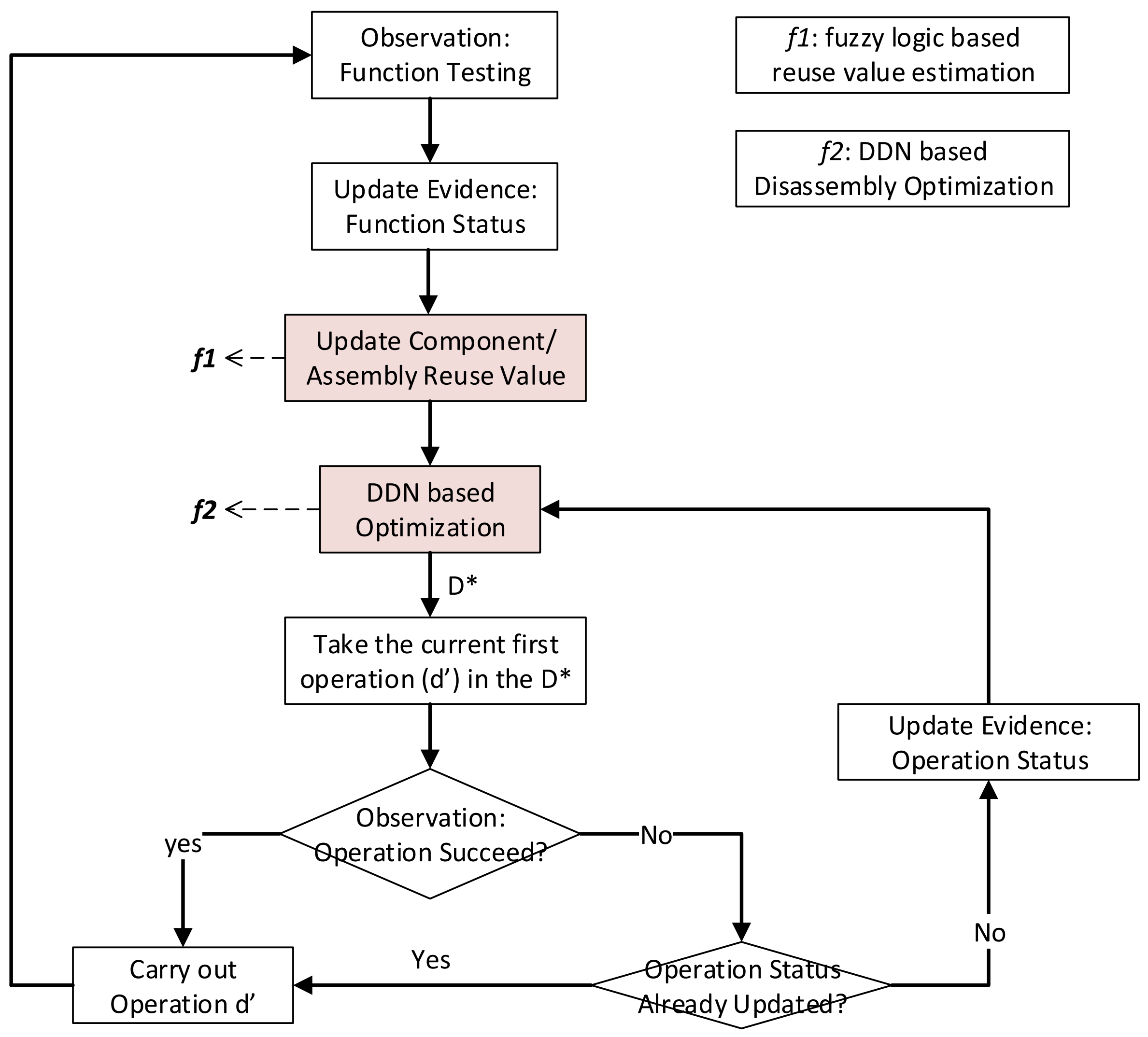

Observation: Function Testing
f1: fuzzy logic based reuse value estimation
f2: DDN based Disassembly Optimization
Update Evidence: Function Status
f1
Update Component/ Assembly Reuse Value
f2
DDN based Optimization
D*
Take the current first operation (d') in the D*
Update Evidence: Operation Status
yes
Observation: Operation Succeed?
No
No
Carry out Operation d'
Yes
Operation Status Already Updated?

Figure 5.5: The Complete Adaptive Disassembly Planning Procedure

The next step is to update the component/assembly reuse value in the DDN using the first sub function (f1: fuzzy logic based reuse value estimation), based on the identified age, market demand and conditional parameter information.

The updated DDN will be sent to the second function (f2: DDN based disassembly optimization), which will generate an optimal disassembly sequence D*, given the currently available identified information.

The disassembly operator will take the first operation (d') suggested in the D* to be the candidate disassembly operation at this stage. Another observation will be carried out to check whether d' can be successfully executed without problems. If d' can be successfully carried out, no updates in DDN are necessary and d' will be physically executed by the disassembly operator, which will yield a new disassembly state. Lastly, the process will be re-routing back to the beginning (function testing) and be applied to the new state.

On the other hand, if d' cannot be successfully carried out, an operation status update (status of d'=fail) is added to the DDN. With the new updated DDN, the optimization process (f2) is carried out again. Two possible scenarios may happen: (1) a new D* will be generated to avoid d' and (2) same D* which insisting on carrying out the original d'. The first scenario is straightforward, which indicates that the cost of executing d', given the evidence the regular operation will fail, is not cost effective and should be avoided. On the other hand, the second scenario indicates that even though d' fails using the regular operation, some special operation (possibly with higher operation cost) should be applied, because the overall utility is still optimal compared to the other options. Lastly, same as before, after carrying out one disassembly operation, either following the original d' or following the new d', a new disassembly state is being reached. The process will re-route back to the function testing step (beginning of the procedure), which will be applied to the new state.

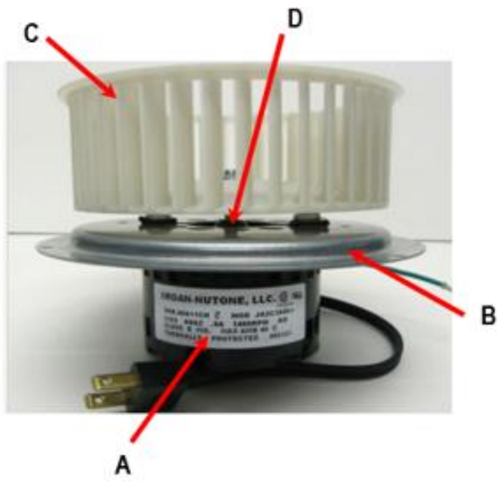
5.4 Case Study

This section verifies the adaptive disassembly planning application using a kitchen exhaust fan assembly. We start with the description of the case study in section 5.4.1. The disassembly decision network for the case study is presented in section 5.4.2. Lastly, the detailed adaptive disassembly decision making process is verified in section 5.4.3.

5.4.1 Description of the Case Study

Figure 5.6 shows the picture of the case study product, which contains four components (A->D) and one assembly (ABCD). Since all of them are associated with a design function, they thus can have a reuse value after being disassembled. We call these types of disassembly objects module here.

On the other hand, three other subassemblies (BCD, BC and BD) exist only in the context of disassembly and they merely represent a stable state in the disassembly process and they don’t have a designed function associated with them (i.e. they are not subassembly in the context of the assembly process). Thus, these type of disassembly objects don’t have a reuse value. The utility related information is listed in table 5.5 and table 5.6 below.



- Disassembly Object as a Module
  - ABCD: Kitchen Exhaust Fan Assembly
  - A: Motor
  - B: Cover
  - C: Blower Wheel
  - D: Rotor Shaft
- Disassembly Object as Stable State
  - BCD
  - BC
  - BD

Figure 5.6: Kitchen Exhaust Fan Assembly

Table 5.5: Utility Information Regarding to the Disassembly Object

Disassembly Object	**Reuse Value**	**Recycle Value**	**Discard Cost**
ABCD	55	N/A	-10
BCD	N/A	N/A	-5
BC	N/A	N/A	-10
BD	N/A	N/A	-5
A	22	N/A	-15
B	10	3	N/A
C	15	2	N/A
D	10	4	N/A

Table 5.6: Utility Information Regarding to the Disassembly Process

Operation	Regular Operation Cost	Special Operation Cost
t1	-5	-10
t2	-8	-16
t3	-5	-20
t4	-10	-15
t5	-8	-15

In table 5.5, there is no reuse value attached to subassembly BC, BD and BCD because they do not have a designed function and they are only valid in the context of disassembly. Also, components B, C and D contain only homogeneous material, and thus they can always be recycled and no discard cost is assigned to them.

Another important piece of information regarding this case study is the process model related to the product, which represents all the feasible disassembly sequences. It is shown in figure 5.7 below, using the petri net as a pictorial notation.

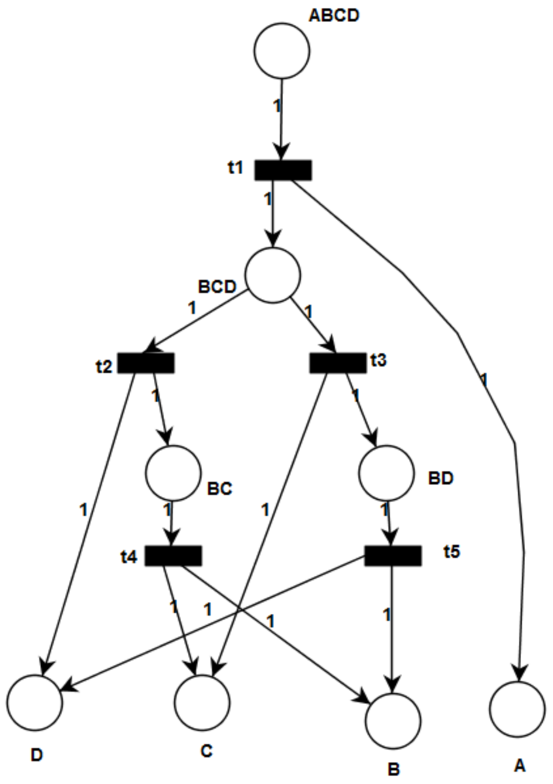


Figure 5.7: The Feasible Disassembly Sequences of the Kitchen Exhaust Fan Assembly

Lastly, the incoming product has two specific uncertainty issues, which are unknown to the disassembly operator in the beginning of the disassembly process:

(1) Blower Wheel is not rotating (i.e. ABCD is not functioning).

(2) Operation t3 cannot be executed in as a regular approach, some special process is needed.

5.4.2 Disassembly Decision Network for the Kitchen Exhaust Fan Assembly

Figure 5.8 shows the disassembly decision network for the kitchen exhaust fan assembly. In order to present it more concisely, the network has been partitioned into different sub models. On the top level, the disassembly decision network contains only two sub models: (1) process model and (2) Bayesian net model. The process model contains the information related to the disassembly object utility (node ABCD_V, A_V, etc.) and several operation sub models. Operation sub model is further an aggregation of the process utility, the process uncertainty and the process decision information. The Bayesian net sub model contains the disassembly object uncertainty information.

The model in the figure 5.8 is a realization of the definition of DDN defined in the section 5.2.1. Five types of nodes and five types of transition arcs are instantiated for the kitchen exhaust fan assembly. Specifically, they are:

Process Decision Node (P-DN): nodes “Operation1”, “Operation2”, etc.

Process Utility Node (P-UTN): nodes “t1_C”, “t2_C”, etc.

Disassembly Object Utility Node (D-UTN): nodes “ABCD_V”, “BCD_V”, “BD_V”, etc.

Process Uncertainty Node (P-UN): nodes “Operation_1_Result”, “Operation_2_Result”, etc.

Disassembly Object Function Uncertainty Node (D-UN): nodes “ABCDFunctionCondition”, “AFunctionCondition”, etc.

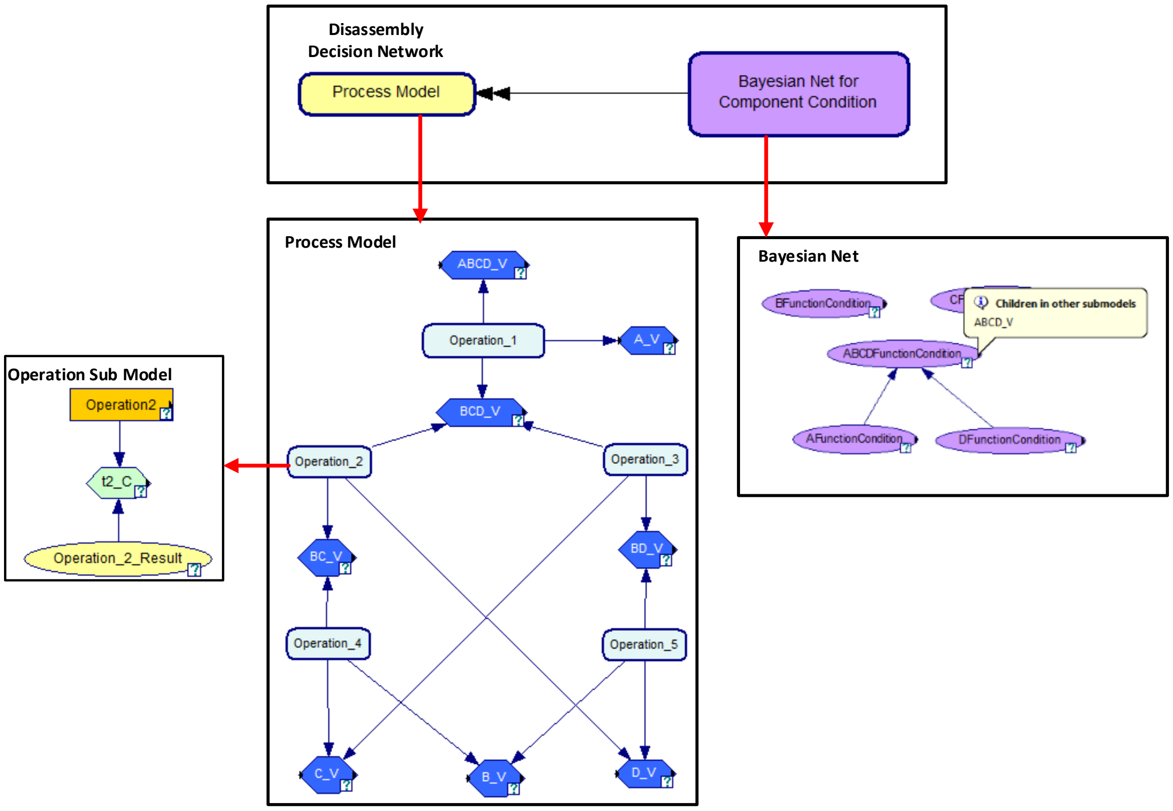


Figure 5.8: The Disassembly Decision Network of the Kitchen Exhaust Fan Assembly

Transition Arc (Type 1: *P-DN* → *P-UTN*): e.g. arc pointing from node “Operation2” to node “t2_C”.

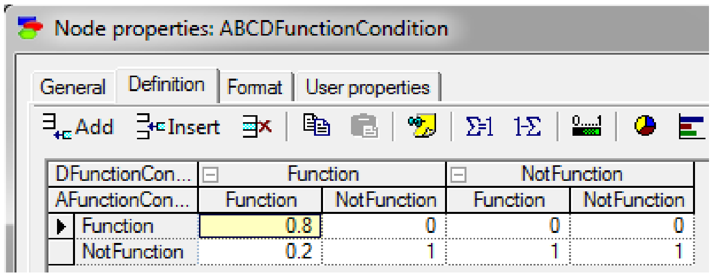
Transition Arc (Type 2: *P-UN* → *P-UTN*): e.g. arc pointing from node “Operation_2_Result” to node “t2_C”.

Transition Arc (Type 3: *D-UN* → *D-UTN):* e.g. arc pointing from node “ABCDFunctionCondition” to node “ABCD_V” (the arc is not shown in figure 5.8).

Transition Arc (Type 4 *P-DN* → *D-UTN*): e.g. arc pointing from node “Operation1” to node “ABCD_V” (the arc is not shown in figure 5.8).

Transition Arc (Type 5 *D-UN →D-UN):* e.g. arc pointing from node “AFunctionCondition” to node “ABCDFunctionCondition”.

Also, conditional probability tables (CPT) are assigned to the relevant nodes. Figure 5.9 shows the user interface to input the CPT for both disassembly object function uncertainty node and process uncertainty node.

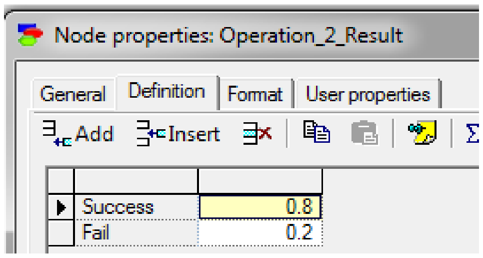


Node properties: ABCDFunctionCondition

General | Definition | Format | User properties

DFunctionCon...	Function		NotFunction	
AFunctionCon...	Function	NotFunction	Function	NotFunction
Function	0.8	0	0	0
NotFunction	0.2	1	1	1

Function Conditional Probability Table for ABCD



Node properties: Operation_2_Result

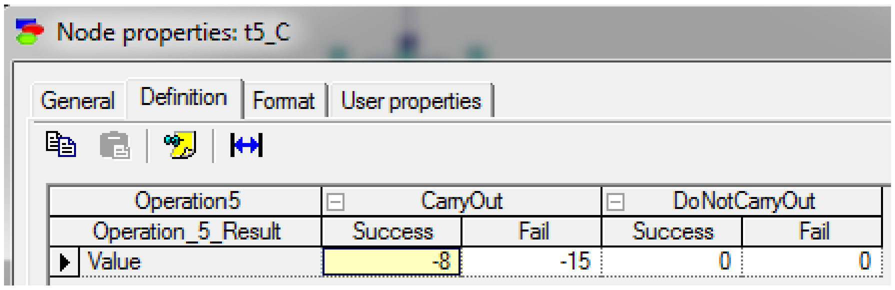
General | Definition | Format | User properties

Success	0.8
Fail	0.2

Process Conditional Probability Table for Operation2

Figure 5.9: An Example Showing the CPT Definition

Lastly, the utility information (both for the disassembly object utility and the process utility) needs to be defined in the DDN. The process utility is relatively straightforward and it is based on whether or not the operation is going to be carried out and whether or not the regular operation will be successful. Figure 5.10 shows an example of the utility definition for operation 5. As it is clear from figure 5.10, the utility (cost) of operation 5 is zero under the condition that operation 5 is not carried out. On the other hand, if operation 5 is to be carried out, the utility (cost) will be either -8 (regular cost) or -15 (special cost), depending on whether or not operation 5 will be executed successfully without a problem.



Node properties: t5_C

General | Definition | Format | User properties

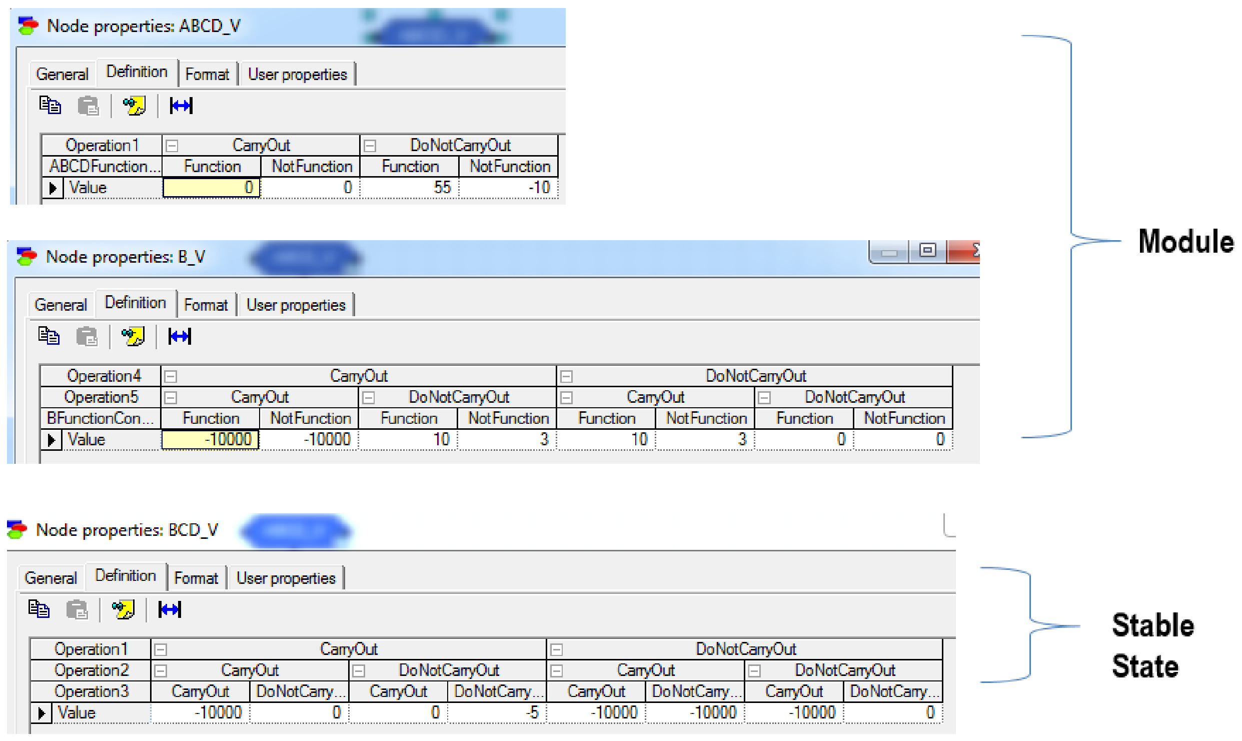
Operation5	CarryOut		DoNotCarryOut	
Operation_5_Result	Success	Fail	Success	Fail
Value	-8	-15	0	0

Figure 5.10: Process Utility Definition Example

The definition of disassembly object utility is classified into two categories: (1) the disassembly object is only a stable state and (2) the disassembly object is a module.

If the disassembly object represents a stable state, like the case for the subassembly BCD (figure 5.11), the only variables influencing the utility are the operation decision nodes pointing to it. If we have 3 influencing operation decision nodes and each of which can take two decision values ("carry out" or "do not carry out"), we can have 2^3=8 possible combinations. Each of the combination will be assigned a utility value (either discard cost or recycling value). Some combination is realistically impossible, such as carrying out operation 1, 2 and 3 in the example in figure 5.11. The utility of such cases will be set to a large negative number (-10000), which insures that it will not be selected as the optimal disassembly path. Some of the combination will yield a zero utility, which means the subassembly is further disassembled into smaller components and thus there is no utility (revenue or cost) associated with that.

A similar mechanism applies to the module type disassembly object, with only one extension: the function uncertainty node has an effect on the utility. If the module is functioning, reuse value could be applied to the utility value, otherwise the discard cost or recycling value will be applied.



Operation1	CarryOut		DoNotCarryOut	
ABCDFunction...	Function	NotFunction	Function	NotFunction
Value	0	0	55	-10

Operation4	CarryOut				DoNotCarryOut			
Operation5	CarryOut		DoNotCarryOut		CarryOut		DoNotCarryOut	
BFunctionCon...	Function	NotFunction	Function	NotFunction	Function	NotFunction	Function	NotFunction
Value	-10000	-10000	10	3	10	3	0	0

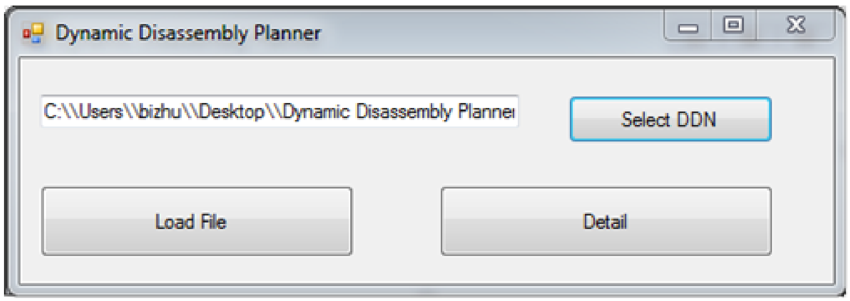
Operation1	CarryOut				DoNotCarryOut			
Operation2	CarryOut		DoNotCarryOut		CarryOut		DoNotCarryOut	
Operation3	CarryOut	DoNotCarry...	CarryOut	DoNotCarry...	CarryOut	DoNotCarry...	CarryOut	DoNotCarry...
Value	-10000	0	0	-5	-10000	-10000	-10000	0

Figure 5.11: Examples Showing the Disassembly Object Utility Definition

5.4.3 Adaptive Sequence Generation for the Kitchen Exhaust Fan Assembly

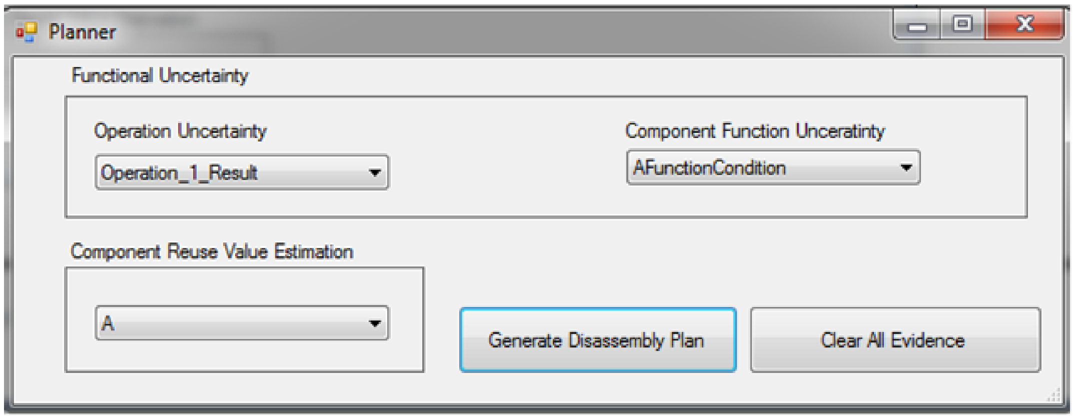
This section verifies the adaptive disassembly sequence generation application using the kitchen exhaust fan assembly. The user interfaces of the developed adaptive disassembly planning application are shown in figure 5.12 below.

Running the application using the kitchen fan assembly by following procedure as defined in figure 5.5, the following adaptive results are generated as shown in table 5.7 below (without component/assembly degradation consideration).



C:\\Users\\bizhu\\Desktop\\Dynamic Disassembly Planner

Loading Disassembly Decision Network

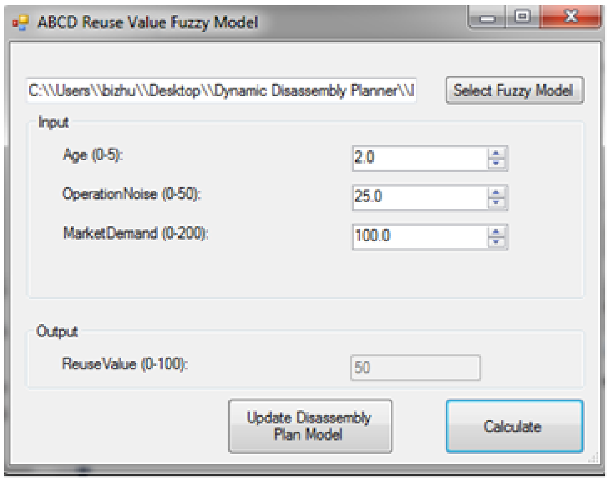


Operation Uncertainty: Operation_1_Result

Component Function Unceratinty: AFunctionCondition

A

Detail User Interface for uncertainty handling and Plan generation



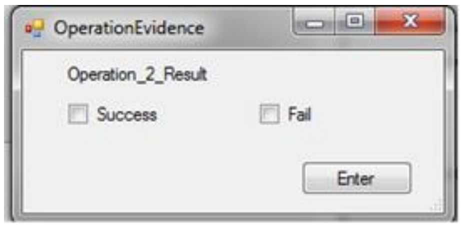
C:\\Users\\bizhu\\Desktop\\Dynamic Disassembly Planner\\

Age (0-5): 2.0

OperationNoise (0-50): 25.0

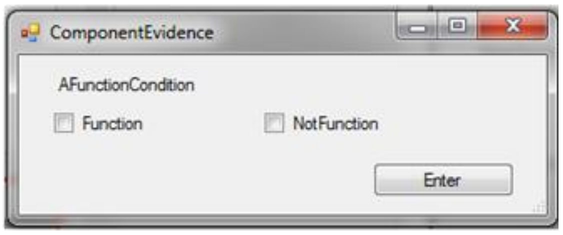
MarketDemand (0-200): 100.0

ReuseValue (0-100): 50



Operation_2_Result

☐ Success ☐ Fail



AFunctionCondition

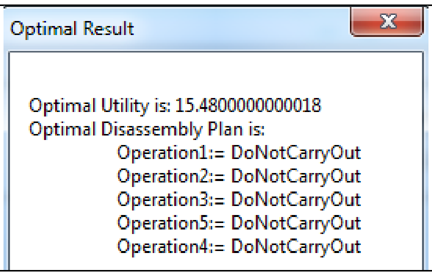
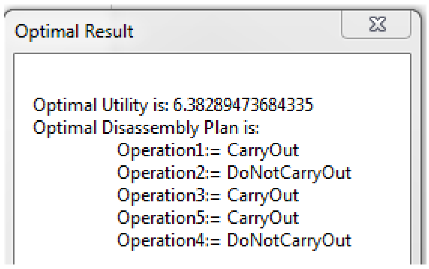
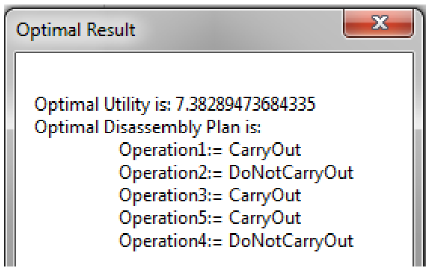
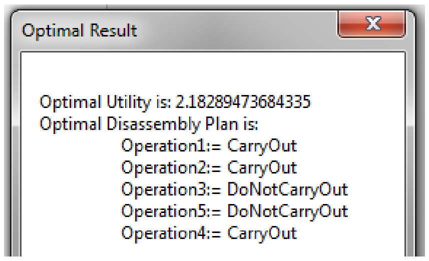
☐ Function ☐ NotFunction

Evidence Updates

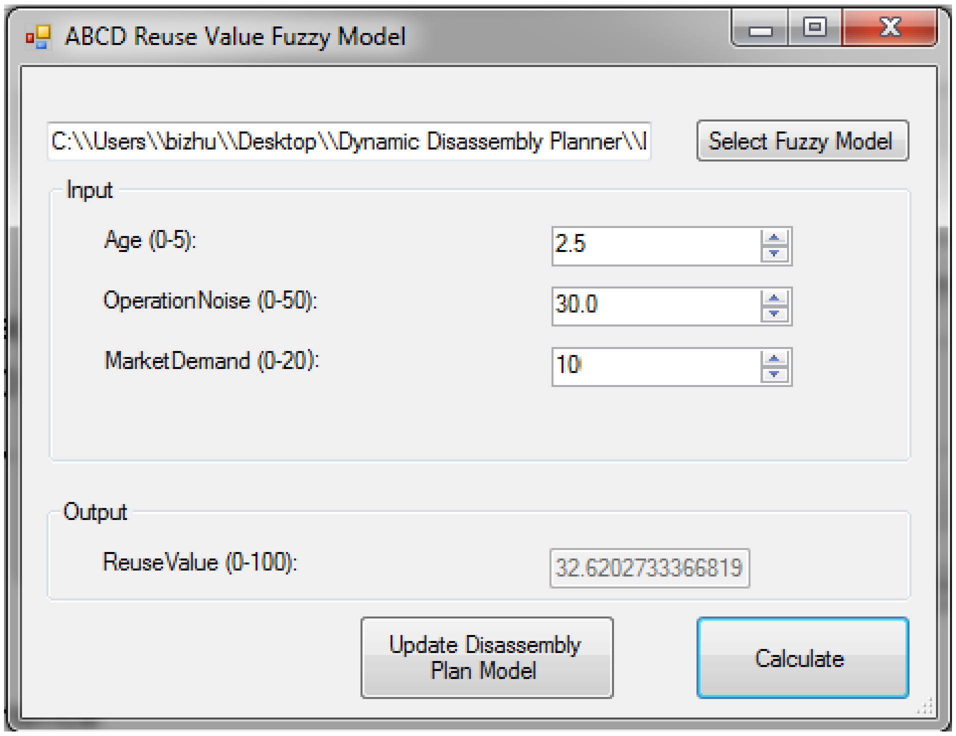
Fuzzy Logic Based Reuse Value Estimation and DDN Updates

Figure 5.12: User Interfaces for the Adaptive Disassembly Planning Application

Table 5.7: Adaptive Disassembly Plan for the Kitchen Fan Assembly

Stage	D*	**Explanation**
Initial Stage (No observation)	 ```Optimal Result``` ```Optimal Utility is: 15.4800000000018``` ```Optimal Disassembly Plan is:``` ```Operation1:= DoNotCarryOut``` ```Operation2:= DoNotCarryOut``` ```Operation3:= DoNotCarryOut``` ```Operation5:= DoNotCarryOut``` ```Operation4:= DoNotCarryOut```	Do not Carry out any disassembly operation, retain the assembly ABCD, which will yield an optimal expected utility 15.48.
Stage 1: Function testing-> ABCD is not functioning	 ```Optimal Result``` ```Optimal Utility is: 6.38289473684335``` ```Optimal Disassembly Plan is:``` ```Operation1:= CarryOut``` ```Operation2:= DoNotCarryOut``` ```Operation3:= CarryOut``` ```Operation5:= CarryOut``` ```Operation4:= DoNotCarryOut```	After function testing, the evident that ABCD is not functioning is updated to the DDN, a new D* is generated, which indicates to carry out operation 1, operation 3 and operation 5. This plan will in the end retrieve component A, B, C and D with a possible expected utility 6.38
Stage 2: Take the current operation in D* (Op1) as candidate and check whether it can be executed successfully -> Op1 can be executed successfully	 ```Optimal Result``` ```Optimal Utility is: 7.38289473684335``` ```Optimal Disassembly Plan is:``` ```Operation1:= CarryOut``` ```Operation2:= DoNotCarryOut``` ```Operation3:= CarryOut``` ```Operation5:= CarryOut``` ```Operation4:= DoNotCarryOut```	Since Op1 can be executed successfully, the generated D* will remain same. However, the expected utility is increasing from 6.38 to 7.38 due to the new evidence.
Stage 2: Take the current operation in D* (Op3) as candidate and check whether it can be executed successfully -> Op3 can't be executed successfully	 ```Optimal Result``` ```Optimal Utility is: 2.18289473684335``` ```Optimal Disassembly Plan is:``` ```Operation1:= CarryOut``` ```Operation2:= CarryOut``` ```Operation3:= DoNotCarryOut``` ```Operation5:= DoNotCarryOut``` ```Operation4:= CarryOut```	The evidence that Op3 can't be executed successfully is updated to the DDN, a new D* is generated. It suggests to carry out operation 1, followed by operation 2 and operation 4, which will avoid the failed operation Op3. The expected utility is 2.18 in this stage.
Both of the two uncertainties have been identified at this point, thus the plan from stage 2 will be the final plan (No change will happen to D* from this stage).		

The application can also handle the component/assembly degradation issues, if assuming subassembly ABCD is functioning. We want to know exactly how much reuse value should be applied for ABCD in the DDN, the fuzzy model for ABCD is going to be used. By observing the crisp values of the input variables (age=2.5 years, market demand=30 units and operation noise=10 decibels), the reuse value of ABCD will be generated (figure 5.13), which indicates a lower value (32.6) compared to the average reuse value (55). This new value will be sent back to update the DDN. The whole process afterwards will be similar to that shown in table 5.7.


ABCD Reuse Value Fuzzy Model

C:\\Users\\bizhu\\Desktop\\Dynamic Disassembly Planner\\I

Input

Age (0-5): 2.5

OperationNoise (0-50): 30.0

MarketDemand (0-20): 10

Output

ReuseValue (0-100): 32.6202733366819

Figure 5.13: Reuse Value estimation for ABCD when Considering Degradation

Chapter 6 CONCLUSION AND FUTURE WORK

This chapter concludes the dissertation and discusses the contributions of this research. In particular, a summary of the Disassembly Information Model (DIM), which establishes the main contribution of this work, is described in section 6.1. In section 6.2, the detailed research issues presented in Chapter 1 are reviewed and how they are being addressed by DIM are discussed. The main research contributions are highlighted in section 6.3. Lastly, possible future directions for extending the work presented in this dissertation are discussed in section 6.4.

6.1 Overview of Disassembly Information Model (DIM)

DIM constitutes a layered information framework designed for multiple applications in the domain of EOL product disassembly planning. DIM is hierarchically structured by layers, which divides the associated Information Models into different levels of abstraction, and thus, separate the general knowledge from the specific knowledge about particular domains and applications. A set of sub models is thus developed and classified into three different layers named the abstract level, the domain level and the application level.

The Information Models in the abstract layer hold fundamental modeling concepts, which is independent of a particular problem or domain, and can therefore be universally applied. They represent the design guidelines (or say design pattern) for the construction of the other sub models in the DIM.

The Information Models in the domain layer capture the knowledge related to a domain of expertise, such as disassembly planning in our case, and they generally don't target at solving a specific problem or task, but rather provides a common foundation for representing a range of different applications. Thus, the Information Model residing on this layer is more specific than those in the abstract layer, but less specific than those in the lower layers.

The Information Models in the application layer targets at modeling the most specific information which is directly usable for a certain application. This dissertation focuses on two disassembly planning applications: (1) Disassembly Sequence Generator and (2) Adaptive Disassembly Planner and the relevant sub models are developed in the application layer of DIM.

The layered structure of DIM reflects the design rationale to reach a balance between information usability and information reusability. These two objectives are conflicting each other in nature: usability implies in depth specialization to meet the requirements of a particular task, whereas reusability requires a certain generality in order to facilitate different applications. Thus, it is difficult to simultaneously achieve a high degree of usability and reusability at the same time. The layered structure of DIM represents a reasonable compromise between information usability and information reusability: models in the higher layers are more abstract and represents reusable information design pattern, whereas models in the lower layers are more specific and can be directly used to solve a specific disassembly planning problem.

DIM is discussed in the form of two complementary parts: (1) an formal DIM description using the UML class diagram and (2) a formal Web Ontology Language (OWL) based DIM implementation. In detail, the developed DIM consists of 11 sub models, comprising of approximately 77 classes, 41 object properties, 14 data properties, 170 major class axioms and 2 semantic web rules. One shortcoming related to the OWL DIM implementation is that few individuals (the instantiated class instances) have been added. We can consider it currently as a lightweight Information Model.

<u>Quality analysis for DIM</u>

Analyzing the quality of an information model is always challenging task and enormous recommendations have been suggested in the literatures. However, there is little consensus being established as a standard. Thus, it is difficult to quantify the degree of quality due to the absence of generally accepted key measures assessing an agreed set of quality indicators. On the other hand, more and more researchers believe that the development of Information Model should follow the “Kaizen” approach, which suggests “continuous improvement during its lifecycle, both in the development stage and the utilization stage”. In other word, Information Model is validated through continuously being used in different applications, and through continuously updating and modifying.

Thus, we believe that a continuous improvement process is inevitable to achieve a good usability-reusability trade-off and thus an Information Model of high quality. In this thesis, we decided to compensate the lack of formal measures by applying DIM to two prototypical software applications. Even if formal measures for quality indicators were available, the degree of (re)usability can be proven ultimately only by testing DIM in a (preferably large) number of different software applications.

6.2 Review of the Research Issues

In this section, the research issues raised in chapter 1 are reviewed and how the developed DIM addresses those issues is discussed.

Q1: What is the information required for disassembly planning and how to model them so that it can be both usable and reusable in the domain of disassembly?

This research question relates to the methodology being utilized for the development of DIM. As mentioned before, a layered IM development methodology is proposed and followed throughout this research work, which provides a reasonable compromise between information reusability and information reusability. The reusability can be shown from the extensive inheritance relationships exists among different sub models residing on the different layers of the DIM, whereas the usability criteria is validated through developing two DIM based disassembly planning related applications.

Q2: How to implement the Disassembly Information Model?

This research question relates to the implementation choice of DIM. In this work, we use the Description Logic (DL) based Web Ontology Language (OWL) for the implementation of DIM. The OWL is developed to support the semantic web applications and it became a World Wide Web Consortium (W3C) recommendation in February 2004. Through using the OWL implementation in two disassembly related applications, we prove that OWL has the full capability to formally and computationally implement the DIM.

Q3: How to validate implemented Disassembly Information Model?

This research question relates to the validation and quality analysis of DIM. Referring to section 5.1 of this dissertation, we compensated the lack of formal quality measures by applying DIM to two prototype software applications. Even if formal measures for quality indicators were available, the degree of (re)usability can be proven ultimately only by testing DIM in a (preferably large) number of different software applications.

6.3 Research Contribution

To the author’s knowledge, this work is the first attempt for the development & utilization of a comprehensive Information Model in the domain of disassembly planning, under the paradigm of sustainable manufacturing. Two major contributions are listed as follows:

- Formal disassembly information representation. Most of the current researches on disassembly modeling are domain and algorithm specific; thus the information is isolated and heterogeneous. That’s why information sharing is difficult. The developed DIM targets on providing a formal, consensual information foundation, which can be promoted to a reference model in the future. This contribution can be broken down into the following aspects:

  - The Generalization of disassembly planning domain information (Product, Process, Uncertainty and Degradation aspect)
  - Development of a layered Information Modeling methodology
  - Implementation of DIM into Web Ontology Language for Machine Processing

- DIM based disassembly planning application modeling. Most of the research on Information Modeling focuses on the development of IM structure, whereas, the application of IM in a real application task is lagging behind. This work fills in this gap by developing two disassembly planning applications based on the extension of DIM: (1) Disassembly Sequence Generator and (2) Adaptive Disassembly Planning. This contribution can be broken down into the following aspects:
  - DIM Extension Mechanism
  - Development of a CAD-API based Disassembly Sequence Generator
  - Development of a Decision Network based Adaptive Disassembly Planner

6.4 Future Work

While this thesis has demonstrated the utilization potentials of applying the DIM in the domain of disassembly planning, many opportunities for extending the scope of this thesis remain. This section presents some of these directions.

From a Light Weight IM to a Reference Model:

The foremost important future work will be to continuously extend and modify the DIM and to upgrade it a reference model (standard). Two parts of work related to this aspect are: (1) the population of more instances (or say individuals) in the DIM and (2) apply DIM to more disassembly planning related applications. The first part of the work will bring DIM to a heavy weight IM and second part of the work will further modify and validates the DIM.

Automated Disassembly:

Another possible direction for the future work is related to the development of the automated disassembly system, which could be a popular research topic. As the reader might notice, there is little information being modeled regarding the disassembly equipment, due to the fact that no standardized equipment for EOL product disassembly are available in practice. However, extending DIM for disassembly equipment is well supported by the current DIM structure and the advantages could be enormous: by relating product with process and finally with equipment, an information loop, from design to realization can be constructed, which will facilitate not only the disassembly decision making, but also equipment level uncertainty handling like equipment reconfiguration planning.

Integrating DIM with Smart Manufacturing infrastructure:

Finally, the last direction of future work focuses on the integration of DIM with the smart manufacturing infrastructure, for better decision making throughout the lifecycle of the product. The scenario is that DIM provide information structure, which is integrated into the LCU system. Data are being collected throughout the lifecycle of the product and can be sent back to the central PLM system for various decision makings, like design suggestion, preventive maintenance, etc.

REFERENCES

Baysal, M. M., Roy, U., Sudarsan, R., Sriram, R. D., & Lyons, K. W. (2004). The open assembly model for the exchange of assembly and tolerance information: Overview and example. Proceedings of 2004 ASME Design Engineering Technical Conferences and Computers and Information in Engineering Conference, 4, 759-770.

Bera, P., Krasnoperova, A., & Wand, Y. (2010). Using ontology languages for conceptual modeling. Journal of Database Management, 21(1), 1-28.

Borst, P., Akkermans, H., & Top, J. (1997). Engineering ontologies. International Journal of Human-Computer Studies, 46(2-3), 365-406.

Chang, X. (2008). Ontology development and utilization in product design (Ph.D.). Available from ProQuest Dissertations & Theses Global. (1020385575).

Chevron, D., Binder, Z., Horacek, P., & Perret, R. (1997). Disassembling process modelling and operations planning under imprecise operation time. Proceedings of 13th IFAC World Congress. L, 367-372.

Clegg, A. J., & Williams, D. J. (1994). The strategic and competitive implications of recycling and design for disassembly in the electronics industry. Proceedings of 1994 IEEE International Symposium On Electronics and The Environment, 6-12.

Cysneiros, L. M., Werneck, V. M., & Kushniruk, A. (2005). Reusable knowledge for satisficing usability requirements. Proceedings of 13th IEEE International Conference on Requirements Engineering, 463-464.

Dassault Systems. (2016). Support and resources for SOLIDWORKS API (application programming interface). Retrieved from https://www.solidworks.com/sw/support/api-support.htm

Dimitris, K., Asbjørn, R., & Bjørn, M. (2008). PROMISE final activity report. Retrieved from http:// http://cordis.europa.eu/

Dong, J., & Arndt, G. (2003). A review of current research on disassembly sequence generation and computer aided design for disassembly. Proceedings of the Institution of Mechanical Engineers, Part B (Journal of Engineering Manufacture), 217, 299-312.

Feng, S. C., Kramer, T., Sriram, R. D., Lee, H., Joung, C. B., & Ghodous, P. (2013). Disassembly process information model for remanufacturing. Journal of Computing and Information Science in Engineering, 13(3), 51-69.

Foufou, S., Fenves, S. J., Bock, C., Rachuri, S., & Sriram, R. D. (2005). A core product model for PLM with an illustrative XML implementation. Proceedings of the International Conference on Product Life Cycle Management. Switzerland: Inderscience Enterprises Limited. 21-32.

Galster, M., & Avgeriou, P. (2012). A variability viewpoint for enterprise software systems. 2012 Joint Working IEEE/IFIP Conference on Software Architecture (WICSA 2012) & European Conference on Software Architecture (ECSA 2012), 267-71.

Gennari, J. H., Musen, M. A., Fergerson, R. W., Grosso, W. E., Crubezy, M., Eriksson, H., Tu, S. W. (2003). The evolution of protégé: An environment for knowledge-based systems development. International Journal of Human-Computer Studies, 58(1), 89-123.

Ghandi, S., & Masehian, E. (2015). Review and taxonomies of assembly and disassembly path planning problems and approaches. Computer-Aided Design, 67–68, 58-86.

Gharfalkar, M., Court, R., Campbell, C., Ali, Z., & Hillier, G. (2015). Analysis of waste hierarchy in the European waste directive 2008/98/EC. Waste Management, 39, 305-313.

Giudice, F. (2010). Disassembly depth distribution for ease of service: A rule-based approach. Journal of Engineering Design, 21(4), 375-411.

Grenchus, E., Keene, R., & Nobs, C. (1997). Demanufacturing of information technology equipment. Proceedings of the 1997 IEEE International Symposium On Electronics and the Environment, 157-160.

Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing. International Journal of Human-Computer Studies, 43(5–6), 907-928.

Gungor, A., & Gupta, S. M. (1999). Issues in environmentally conscious manufacturing and product recovery: A survey. Computers & Industrial Engineering, 36(4), 811-853.

Gutierrez, C., Garbajosa, J., Diaz, J., & Yague, A. (2013). Providing a consensus definition for the term "smart product". 2013 20th IEEE International Conference and Workshops on The Engineering of Computer Based Systems (ECBS), 203-211.

Halpin, T. A. (2001). Information modeling and relational databases: From conceptual analysis to logical design, Morgan Kaufmann. ISBN-13: 978-1558606722

Hamidullah, Bohez, E., & Irfan, M. A. (2006). Assembly features: Definition, classification, and instantiation. 2nd Annual International Conference on Emerging Technologies 2006, ICET 2006, November 13, 2006 - November 14, 617-623.

Homem de Mello, L. S., & Sanderson, A. C. (1989). A correct and complete algorithm for the generation of mechanical assembly sequences. Proceedings of IEEE International Conference On Robotics and Automation, 1, 56-61.

Hsin-Hao, H., Wang, M. H., & Johnson, M. R. (2000). Disassembly sequence generation using a neural network approach. Journal of Manufacturing Systems, 19(2), 73-82.

IEEE standard glossary of software engineering terminology. (1983). (ANSI/IEEE Standard, No. 729). USA: Inst. Electrical & Electronics. Engineering, USA.

Ilgin, M. A., & Gupta, S. M. (2010). Environmentally conscious manufacturing and product recovery (ECMPRO): A review of the state of the art. Journal of Environmental Management, 91(3), 563-591.

Kanai, S., Sasaki, R., & Kishinami, T. (1999). Graph-based information modeling of product-process interactions for disassembly and recycle planning. Environmentally Conscious Design and Inverse Manufacturing, Proceedings of First International Symposium On EcoDesign, 772-777.

Kim, K., Manley, D. G., & Yang, H. (2006). Ontology-based assembly design and information sharing for collaborative product development. Computer-Aided Design, 38(12), 1233-1250.

Kiritsis, D. (2011). Closed-loop PLM for intelligent products in the era of the internet of things. Computer-Aided Design, 43(5), 479-501.

Klinker, G., Bhola, C., Dallemagne, G., Marques, D., & McDermott, J. (1991). Usable and reusable programming constructs. Knowledge Acquisition, 3(2), 117-135.

Kumar, P. P. (2008). Design process modeling: Towards an ontology of engineering design activities (M.S.). Available from ProQuest Dissertations & Theses Global. (304676073).

Kutz, O., & Garbacz, P. (2014). Formal ontology in information systems. Proceedings of FOIS'2014. Amsterdam: IOS Press, 3-15.

Kwaan, J. R. (1994). Bayesian belief networks for industrial applications. Proceedings of Adaptive Computing and Information Processing, 2, 625-640.

Lambert, A. J. D. (Fred) and Surendra M. Gupta. (2005). Disassembly modeling for assembly, maintenance, reuse, and recycling. United States: CRC Press.

Lambert, A. J. D. (1997). Optimal disassembly of complex products. International Journal of Production Research, 35(9), 2509-2524.

Lemaignan, S., Siadat, A., Dantan, J., and Semenenko, A. (2006). MASON: A proposal for an ontology of manufacturing domain. IEEE Workshop On Distributed Intelligent Systems: Collective Intelligence and its Applications, 195-200.

Mann, C. J. H. (2003). The description logic handbook - theory, implementation and applications. Cambridge University Press, ISBN-13: 978-0521150118.

Memon, N., Ortiz-Arroyo, D., & Larsen, H. L. (2005). Analysis of state of art in semantic web languages - an overview. Mehran University Research Journal of Engineering and Technology, 24(3), 197-210.

Millar, S. A. (2005). E-waste legislation. Paper, Film and Foil Converter, 79(5), 16-18.

Moore, K. E., Gungor, A., & Gupta, S. M. (1998). A petri net approach to disassembly process planning. Computers & Industrial Engineering, 35(1–2), 165-168.

Murayama, T., Oba, F., Abe, S., & Yamamichi, Y. (2001). Disassembly sequence generation using information entropy and heuristics for component replacement. Proceedings of IEEE International Symposium On Assembly and Task Planning, 208-213.

PENEV, K. D., PENEV, K. D., & de RON, A. J. (02). Determination of a disassembly strategy. International Journal of Production Research, 34(2), 495-506.

Ridder, C., & Scheidt, L. (1998). Practical experience in the Sony disassemly evaluation workshop. Proceedings of the 1998 IEEE International Symposium On Electronics and the Environment, 94-98.

Um, J., Joo-Sung Yoon, & Suk-Hwan Suh. (2008). An architecture design with data model for product recovery management systems. Resources, Conservation & Recycling, 52(10), 1175-84.

Uschold, M., & Gruninger, M. (1996). Ontologies: Principles, methods and applications. The Knowledge Engineering Review, 11(02), 93-136.

Vukadinovic, D. (2013). Fuzzy logic: Applications, systems, and technologies. Hauppauge, New York: Nova Science Publishers, Inc. ISBN: 978-1-62417-151-2.

Wang, L., Wang, X. V., Gao, L., and Váncza, J. (2014). A cloud-based approach for WEEE remanufacturing. CIRP Annals - Manufacturing Technology, 63(1), 409-412.

Xia, K., Gao, L., Wang, L., & Li, W. (2015). A semantic information services framework for sustainable WEEE management toward cloud-based remanufacturing. Journal of Manufacturing Science and Engineering, 137(6), 11-22.

Zhang, H. C., & Kuo, T. C. (1996). A graph-based approach to disassembly model for end-of-life product recycling. 19th IEEE/CPMT Symposium on Electronics Manufacturing Technology, 247-254.

Zhu, B., Sarigecili, M. I., & Roy, U. (2013). Disassembly information model incorporating dynamic capabilities for disassembly sequence generation. Robotics and Computer-Integrated Manufacturing, 29(5), 396-409.

Ziout, A. (2013). Innovative design for active disassembly and sustainable product recovery (Ph.D.). Available from ProQuest Dissertations & Theses Global. (1465052255).

Zuniga, G. L. (2001). Ontology: Its transformation from philosophy to information systems. Proceedings of FOIS'01: 2nd International Conference on Formal Ontology in Information Systems, 187-97.

Zussman, E., & Meng Chu Zhou. (2000). Design and implementation of an adaptive process planner for disassembly processes. IEEE Transactions on Robotics and Automation, 16(2), 171-179.

Zussman, E., MengChu Zhou, & Caudill, R. (1998). Disassembly petri net approach to modeling and planning disassembly processes of electronic products. Proceedings of the 1998 IEEE International Symposium On Electronics and the Environment, 331-336.

APPENDIX

Summary of DIM Model

Model	Imported Model	Class	Class Axioms	Object Property	Datatype Property
N-ary-relationship.owl	N/A	Object		involves	relationAttribute
		Relationship	involves **min** 2 Object	hasOrigin	
		Directed Relationship	Subclass of Relationship	hasTarget	
			hasOrigin **exactly** 1 Object		
			hasTarget **some** Object		
Part_whole.owl	N-ary-relationship.owl	Part	Subclass of Object	hasPart	
		Whole	hasPart **some** Part		
			Subclass of Object		
Graph.owl	Part_whole.owl	Object	hasConnector **some** Connector	hasConnectingPoint	
		Arc	hasConnectingPoint **exactly** 2 ConnectingPoint	hasConnector	
			isDirectlyConnectedTo **exactly** 2 Node		
			Subclass of Object		
		Connector	isDirectlyConnectedTo **exactly** 1 Connector	hasPort	
			Subclass of Object		
			Subclass of Relationship		
		Connecting Point	isDirectlyConnectedTo **exactly** 1 Port	isDirectlyConnectedTo	
			Subclass of Connector		
		Port	isDirectlyConnectedTo **exactly** 1 ConnectingPoint		
			Subclass of Connector		
		Node	hasPort **some** Port		
			isDirectlyConnectedTo **some** Arc		
			Subclass of Object		
System.owl	Part_whole.owl	Aspect	Subclass of Object	contains	
		System	Subclass of Object	isConsideredUnderAspectOf	
		Atomic SubSystem	Equivalent To: System **and** (contains **exactly** 0 AtomicSubSystem)	isModeledBy	
			Subclass of System		
		Aspect System	Equivalent To: AtomicSubSystem **and** (isConsideredUnderAspectOf **exactly** 1 Aspect)	models	
			Subclass of AtomicSubSystem		
		Composite SubSystem	Equivalent To: System **and** (contains **some** AtomicSubSystem)		
			Subclass of System		

Summary of DIM Model Continued

Model	Imported Model	Class	Class Axioms	Object Property	Datatype Property
System.owl	Part_whole.owl	Model	models exactly 1 System		
			Subclass of System		
DisassemblyPlanning System.owl	System.owl	Disassembly PlanningSystem	Subclass of CompositeSubSystem		
	Product.owl		contains **exactly** 1 Product		
	Process.owl		contains **exactly** 1 Process		
	Degradation.owl		contains **exactly** 1 Uncertainty		
	Uncertainty.owl		contains **exactly** 1 Degradation		
Product.owl	System.owl	Structure	Subclass of Aspect	belongsTo	
		Component	containsMaterial **some** Material	contains	
			hasConstraintFeature **some** ConstrainingFeature		
			isDirectlyConnectedTo **some** ComponentContact		
			Subclass of Feature		
		Connecting Component	Equivalent To: Component And (hasDegreeOfFreedom **some** DegreeOfFreedom)	containsMaterial	
			belongsTo **exactly** 1 Connection		
			Subclass of Component		
		Fastener	Subclass of ConnectingComponent	hasComponent	
		VirtualConnecting Component	Equivalent To: ConnectingComponent **and** (**not** (Fastener))	hasConnectingPoint	
			Subclass of ConnectingComponent		
		OrdinaryComponent	Equivalent To: OrdinaryComponent **and** (hasDegreeOfFreedom **exactly** 0 DegreeOfFreedom)	hasConstraintFeature	
			Subclass of Component		
		AtomicOrdinary Component	Equivalent To: OrdinaryComponent **and** (hasComponent **exactly** 1 Component)	hasDegreeOfFreedom	
			Subclass of Component		
		CompositeAtomic OrdinaryComponent	Equivalent To: AtomicOrdinaryComponent **and** (containsMaterial **min** 2 Material)	hasSubAssembly	
			Subclass of AtomicOrdinaryComponent		
		HomogeneousAtomic OrdinaryComponent	Equivalent To: AtomicOrdinaryComponent **and** (containsMaterial **exactly** 1 Material)	isDirectlyConnectedTo	
			Subclass of AtomicOrdinaryComponent		

Summary of DIM Model Continued

Model	Imported Model	Class	Class Axioms	Object Property	Datatype Property
Product.owl	System.owl	ComplexOrdinary Component	Equivalent To: OrdinaryComponent **and** (hasComponent **min** 2 Component)		
			Subclass of OrdinaryComponent		
		CompositeComplex OrdinaryComponent	Equivalent To: ComplexOrdinaryComponent **and** (containsMaterial **min** 2 Material)		
			Subclass of ComplexOrdinaryComponent		
		HomogeneousComplex OrdinaryComponent	Equivalent To: ComplexOrdinaryComponent **and** (containsMaterial **exactly** 1 Material)		
			Subclass of ComplexOrdinaryComponent		
		ComponentContact	hasConnectingPoint **exactly** 2 ConnectingInterface		
			isDirectlyConnectedTo **exactly** 2 Component		
			Subclass of Object		
		ConnectingInterface	isDirectlyConnectedTo **exactly** 1 ConstrainingFeature		
			Subclass of Object		
		Connection	contains **some** ConnectingComponent		
			Subclass of Object		
		ConstrainingFeature	belongsTo **exactly** 1 Component		
			isDirectlyConnectedTo **exactly** 1 ConnectingInterface		
			Subclass of Object		
		DegreeOfFreedom	Subclass of Object		
		Material	Subclass of Object		
		SubAssembly	hasComponent **min** 2 Component		
			Subclass of Object		
		Product	Equivalent To: AspectSystem **and** (isConsideredUnderAspectOf **exactly** 1 Structure)		
			Subclass of AspectSystem		
			hasComponent **some** Component		
			hasSubAssembly **some** SubAssembly		
Process.owl	System.owl	Behavior	Subclass of Aspect	breaks	normalCost
		DisassemblyObject	Subclass of Object	creates	specialCost
		Component	Subclass of DisassemblyObject	hasAction	
		Product	Subclass of DisassemblyObject	hasOperation	
		SubAssembly	Subclass of DisassemblyObject		

Summary of DIM Model Continued

Model	**Imported Model**	**Class**	**Class Axioms**	**Object Property**	**Datatype Property**
Process.owl	System.owl	Process	Equivalent To: AspectSystem **and** (isConsideredUnderAspectOf **exactly** 1 Behavior)		
			Subclass of AspectSystem		
			breaks **exactly** 1 DisassemblyObject		
			creates **min** 2 DisassemblyObject		
			Subclass of DirectedRelationship		
			normalCost **some** double		
			specialCost **some** double		
		Action	Subclass of Process		
		Operation	Subclass of Process		
			hasAction some Action		
		Task	Subclass of Process		
			hasOperation **some** Operation		
Uncertainty.owl	System.owl	Disturbance	Subclass of Aspect	contains	
		ConditionalProbabilityTable	Subclass of Object	functionalDepends	
		FunctionFailure ProbabilityTable	Subclass of ConditionalProbabilityTable	relatesTo	
		ProcessSuccess ProbabilityTable	Subclass of ConditionalProbabilityTable		
		DisassemblyObject	contains **exactly** 1 FunctionFailureProbabilityTable		
			Subclass of Object		
		Component	Subclass of DisassemblyObject		
		Product	Subclass of DisassemblyObject		
		SubAssembly	Subclass of DisassemblyObject		
		Process	contains exactly 1 ProcessSuccessProbabilityTable		
			Subclass of Object		
		Uncertainty	AspectSystem **and** (isConsideredUnderAspectOf **min** 1 Disturbance)		
			Subclass of AspectSystem		

Summary of DIM Model Continued

Model	Imported Model	Class	Class Axioms	Object Property	Datatype Property
Degradation.owl	System.owl	DisassemblyObject	hasAge **exactly** 1 Age	hasAge	functionType
		DisassemblyObject FuzzyTerm	hasConditionParameter **some** ConditionParameter	hasConditionParameter	lowerLimit
			hasMarketDemand **exactly** 1 MarketDemand	hasFuzzyTerm	name
			hasReuseValue **exactly** 1 ReuseValue	hasMarketDemand	parameter
			hasRuleSet **some** RuleSet	hasReuseValue	upperLimit
			Subclass of Object	hasRuleSet	variableType
			functionType **exactly** 1 string	relatesTo	
		FuzzyTerm FuzzyVariable	name **exactly** 1 string		
			Subclass of Object		
			parameter **exactly** 1 string		
			hasFuzzyTerm **some** FuzzyTerm		
			lowerLimit **exactly** 1 double		
			Subclass of Object		
			upperLimit **exactly** 1 double		
			variableType **exactly** 1 string		
		Age	Subclass of FuzzyVariable		
		ConditionParameter	Subclass of FuzzyVariable		
		MarketDemand	Subclass of FuzzyVariable		
		ReuseValue	Subclass of FuzzyVariable		
		RuleSet	Subclass of Object		
		Degradation	Subclass of AspectSystem		
		Degradation	relatesTo **some** DisassemblyObject		
			relatesTo **some** DisassemblyObject		

Summary of DIM Model Continued

Model	**Imported Model**	**Class**	**Class Axioms**	**Object Property**	**Datatype Property**
DisassemblySequence Generator.owl	Product.owl	Constraining FeaturePair	target **exactly** 1 ConstrainingFeature	target	reuseValue
			direction **exactly** 1 string	belongsTo	recycleValue
		Component	discardCost **exactly** 1 double	hasContactLoop	discardCost
			recycleValue **exactly** 1 double		direction
			reuseValue **exactly** 1 double		
		Constraining Feature	belongsTo **some** ConstrainingFeaturePair		
		ContactLoop	Subclass of SubAssembly		
			discardCost **exactly** 1 double		
			recycleValue **exactly** 1 double		
			reuseValue **exactly** 1 double		
		ContactLoop Cluster	Subclass of SubAssembly		
			discardCost **exactly** 1 double		
			recycleValue **exactly** 1 double		
			reuseValue **exactly** 1 double		
			hasContactLoop **min** 2 ContactLoop		
Adaptive Disassembly Planning.owl	Process.owl	Disassembly_Object_ Utility_Node	relatesTo **exactly** 1 (Process model: DisassemblyObject)	relatesTo	
	Degradation .owl		relatesTo **exactly** 1 (Degradation model: DisassemblyObject)		
	Uncertainty .owl	Disassembly_Object_ Function_Uncertainty _Node	influence **exactly** 1 Disassembly_Object_Utility_Node	influence	
			relatesTo **exactly** 1 (Uncertainty model: DisassemblyObject)		
		Process_Uncertainty_Node	influence exactly 1 Process_Utility_Node		
			relatesTo **exactly** 1 (Uncertainty model: Process)		
		Process_Utility_Node	relatesTo **exactly** 1 (Process model: Process)		
		Process_Decision_Node	influence **exactly** 1 Process_Utility_Node		

BIBLIOGRAPHY

PERSONAL INFORMATION

Name: Bicheng Zhu
Email: bizhu@syr.edu
Contact: 315-416-1643
Address: Link 277 Syracuse University,
Syracuse, NY 13244

EDUCATION

12/2012- 6/2016
L.C. Smith College of Engineering & Computer Science, Syracuse University,
Doctor of Philosophy in Mechanical and Aerospace engineering

8/2010- 12/2012
L.C. Smith College of Engineering & Computer Science, Syracuse University,
Master of Science in Mechanical and Aerospace engineering

9/2003- 7/2007
School of Mechanical Engineering, Shanghai Jiao Tong University
Bachelor of Science in Thermal Energy and Power Engineering (Vehicle Engine)
Bachelor of Science in Computer Technology and Application

WORK & INTERN EXPERIENCE

9/1/2015-Present
Research Assistantship at L.C. Smith College of Engineering & Computer Science
Topic: Computational Platform for Bio-Product Design

12/2012 – 05/15/2015
Teaching Assistant at L.C. Smith College of Engineering & Computer Science
Teaching and tutoring in courses/labs:

- MAE 184: Engineering Graphics
- MEE 571: Computer Aided Design
- MAE 321: Manufacturing Processes
- MAE 635: Advanced Mechanics of materials

9/ 2007—7/2010
Structural Engineer at Shanghai Koito Automotive Lamp Co., Ltd
Major Designed Model:

- SGM 308 NGS CAR-Tail light kits
- SGM 308 NGS CAR- High mount stop lamp
- GMX 353 Tail light kits
- SGM 618 (Excelle) Tail light kits
- B61 Tail light kits

12/ 2006 –6/2007
Intern at Saint-Gobain Automobile Glass Company

7/ 2006 –9/2006
Intern at Shanghai Maple Engine Co., Ltd

ACADEMIC PUBLICATIONS

1. **Zhu, B**., Mehmet I. S., Roy, U., Disassembly information model incorporating dynamic capabilities for disassembly sequence generation, Robotic sand Computer Integrated Manufacturing, v 29, n 5, p 396-409, Oct. 2013
2. Mehmet I. Sarigecili*, Mehmet Murat Baysal, **Bicheng Zhu** and Utpal Roy, A disassembly process model for end-of-life (EOL) activities of manufactured products, International Journal of Sustainable Manufacturing, Vol. 3, No. 1, p37-56, 2013
3. Zhu, B., Roy, U., Ontology-based disassembly information system for enhancing disassembly planning and design, Int J Adv Manuf Technol., DOI 10.1007/s00170-014-6704-8, 2015
4. Heng Zhang, **Bicheng Zhu**, Yunpeng Li, Omer Yaman and Utpal Roy, Development and Utilization of A Process-oriented Information Model for Sustainable Manufacturing, Journal of Manufacturing System, doi:10.1016/j.jmsy.2015.05.003.
5. **Zhu, B**., Mehmet I. S., Roy, U., Disassembly information model incorporating dynamic capabilities for disassembly sequence generation, GSCM 2012 conference, Turkey, 2012
6. Mehmet I. Sarigecili*, Mehmet Murat Baysal, **Bicheng Zhu** and Utpal Roy, A disassembly process model for end-of-life (EOL) activities of manufactured products, GSCM 2012 conference, Turkey, 2012
7. **Bicheng Zhu** and Utpal Roy, Uncertain Information Representation And Its Usage In Disassembly Modeling, Proceedings of the ASME Design Engineering Technical Conference, v 2 A, 2013, ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, IDETC/CIE 2013
8. Utpal Roy, **Bicheng Zhu**, Denial Rice, The Information Framework for Material Behavior Representation, IEEE International Conference on Automation Science and Engineering, p 380-385, 2013

9. **Bicheng Zhu**, Utpal Roy, Towards A Semantic Web Approach for Disassembly Planning, Proceedings of ASME 2014 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, IDETC/CIE 2014, Buffalo (Accept)
10. Utpal Roy, **Bicheng Zhu**, Development of Material Information Model for the Injection Molding Process and Product, Proceedings of ASME 2014 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, IDETC/CIE 2014, Buffalo
11. Omer Yaman, **Bicheng Zhu**, and Utpal Roy, Towards The Development Of An Ontology-Based Product Requirement Model, Proceedings of the ASME 2014 International Mechanical Engineering Congress and Exposition & Computers and Information in Engineering Conference IMECE/CIE 2014
12. Utpal Roy, **Bicheng Zhu**, Yunpeng Li, Heng Zhang and Omer Yaman, Mining Big Data In Manufacturing: Requirement Analysis, Tools And Techniques, Proceedings of the ASME 2014 International Mechanical Engineering Congress and Exposition & Computers and Information in Engineering Conference IMECE/CIE 2014
13. Utpal Roy, Yunpeng Li and **Bicheng Zhu,** Building a Rigorous Foundation for Performance Assurance Assessment Techniques for "Smart" Manufacturing Systems, 2014 IEEE International Conference on Big Data, Washington DC, USA, Oct

RESEARCH FOCUS

- Information Modeling
- Ontology Engineering
- Product Disassembly
- Sustainable Manufacturing
- PLM System